PRIME AND PDQ SORTS EFFICIENT MINIMAL
STORAGE SORTING ALGORITHMS

Final Report

R. Hilbrand

August 1979

Approved for public release; distribution unlimited; statement applied August 1979.

Prepared for

THE JOINT LOGISTICS COMMANDERS
JOINT TECHNICAL COORDINATING GROUP ON
AIRCRAFT SURVIVABILITY

79 09 5 027
FOREWORD

This report summarizes the results of research performed by the Aeronautical Systems Division, Wright-Patterson AFB, OH. The work was performed between November 1976 and March 1978, and the Project Engineer for this effort was G. B. Bennett.

The work was sponsored by the JTCG/AS as part of the 3-year TEAS (Test and Evaluation Aircraft Survivability) program. The TEAS program was funded by DDR&E/ODDT&E. The effort was conducted under the direction of the JTCG/AS Vulnerability Assessment Subgroup, as part of JTCG/AS Work Unit VA-6-02F, Development of Aircraft Preliminary Design Assessment Methodology.

This report presents and summarizes two sorting algorithms, PRIME Sort and PDQ Sort, developed in the Aeronautical Systems Division Computer Science Center in support of vulnerability assessment computer programs in use by the Deputy for Development Planning.

NOTE

This technical report was prepared by the Vulnerability Assessment Subgroup of the Joint Technical Coordinating Group on Aircraft Survivability in the Joint Logistics Commanders' organization. Because the Services' aircraft survivability development programs are dynamic and changing, this report represents the best data available to the subgroup at this time. It has been coordinated and approved at the JTCG subgroup level. The purpose of the report is to exchange data on all aircraft survivability programs, thereby promoting interservice awareness of the DoD aircraft survivability program under the cognizance of the Joint Logistics Commanders. By careful analysis of the data in this report, personnel with expertise in the aircraft survivability area should be better able to determine technical voids and areas of potential duplication or proliferation.
**REPORT DOCUMENTATION PAGE**

<table>
<thead>
<tr>
<th>REPORT NUMBER</th>
<th>JTCG/AS78-V-004</th>
</tr>
</thead>
<tbody>
<tr>
<td>TITLE (and Subtitle)</td>
<td>PRIME &amp; PDQ Sorts - Efficient Minimal Storage Sorting Algorithms</td>
</tr>
<tr>
<td>AUTHOR(s)</td>
<td>Roy R. Hilbrand</td>
</tr>
<tr>
<td>PERFORMING ORGANIZATION NAME AND ADDRESS</td>
<td>Aeronautical Systems Division, ASD/ADDP, Wright-Patterson AFB, Ohio 45433</td>
</tr>
<tr>
<td>REPORT DATE</td>
<td>Aug 79</td>
</tr>
<tr>
<td>NUMBER OF PAGES</td>
<td>54</td>
</tr>
</tbody>
</table>

**ABSTRACT**

(See reverse side)

79 09 5 027
Aeronautical Systems Division

PRIME and PDQ Sorts — Efficient Minimal Storage Sorting Algorithms, by R. R. Hilbrand, Wright-Patterson AFB, OH, ASD, for Joint Technical Coordinating Group/Aircraft Survivability, August 1979. 54 pp. (JTCG/AS-78-V-004, publication UNCLASSIFIED.)

One of the problems involved in computer programs for vulnerability assessment is that of rapidly sorting and arranging large sets of data. Two sorting algorithms, designated PRIME and PDQ, have been developed at ASD to more efficiently perform this function in vulnerability programs such as SESTEM and FASTGEN II. The results are compared to those obtained with three other algorithms, SHELLSORT, TREESORT3, and SINGLETON. The newly developed sorts are shown to be significantly faster on the ASD CDC 6600 computer than the existing sorts. When used in an ASD missile endgame model SESTEM, the average run time was reduced by 20 to 25%. Program listings, flow charts, and typical output data are presented.
CONTENTS

Introduction .................................................................................. 1

Program Development .................................................................... 1
  Background .................................................................................. 1
  Partial Sorting ............................................................................ 2
  PDQ Algorithm ........................................................................... 2

Sort Code Testing Algorithm .......................................................... 2
  Distributions .............................................................................. 2
  Testing Environment .................................................................. 3
    Machine Dependancy ............................................................... 3
    Variations ............................................................................... 8

Prime Sorting Algorithm ............................................................... 14

Performance Evaluation ................................................................. 22
INTRODUCTION

One of the steps in most automated vulnerability assessments involves the rapid sorting of large strings of data. In the ASD (Aeronautical Systems Division) missile endgame model SESTEM,\(^1\) for example, the aircraft components struck by the expanding fragment spray-band are calculated and stored, and then must be sorted. The sorted data are then used to compute the components struck in order of time intercepted and the resulting aircraft probability of kill. Similar data string storage and sort problems are involved in other vulnerability assessment computer programs such as FASTGEN II.\(^2\) In SESTEM, about 40% of the computation time is typically involved with the sorting and fragment vulnerability computations.

For these reasons, the development of efficient algorithms to sort these strings of data are of considerable importance. Since large portions of the total computer run times are typically involved in performing this sorting, any improvements in them will show direct payoffs in terms of decreased run times, more rapid turnaround, greater program efficiencies, and decreased costs. The sorting algorithms in use in the vulnerability assessment programs were evaluated, and two new and more efficient programs were written. When the sorting algorithm was inserted in a new version of the SESTEM program, the average time for a run was decreased by 20 to 25%. These programs, and comparisons with some existing programs are presented in this report.

PROGRAM DEVELOPMENT

BACKGROUND

The availability of efficient general purpose sort algorithms and the sharp reduction of cost per computation of present generation computers has reduced interest in research into sort algorithms from earlier levels. Aside from the intrinsic challenge presented by the sorting problem, optimization of existing and developing applications programs written in FORTRAN IV, or similar level languages, indicate a still existing need for compact, efficient, in-line sort algorithms that are readily adaptable to specialized ends.

One approach to meeting this need is to devise a partial sort, an efficient algorithm by which a string of numbers is nearly sorted; and complete the sort by a method such as binary insertion, which can take advantage of a high degree of order in a number string.

---

\(^1\)Aeronautical Systems Division. "SESTEM Missile Endgame Model", by G.B. Bennett, Wright-Patterson AFB, OH, ASD. September 1977. (ASD/XRH memo.)

\(^2\)Aeronautical Systems Division. FASTGEN II Target Description Computer Program, by D. Cudney, Wright-Patterson AFB, OH, ASD. March 1978. (Report ASD-TR-77-24.)
PARTIAL SORTING

The basic strategy in producing partially sorted strings is to successively partition a set of numbers by exchange comparisons. Let NUM be a string of numbers of length NO equal to an integral power of two. The set of NO numbers is divided into two subsets of equal size, and the elements of one subset are compared (and exchanged when necessary) to the elements of the other subset in such a way that every element is involved in a comparison once only. As a worst case, after any comparison of sets as described, 25% of the elements less than or equal to the median value will be located below the median. Successive set divisions and comparisons continue the process, removing extreme values from the middle of the array and enabling further distribution to take place, until there are NO subsets of 1 element each. If the NO elements are distinct and represented by 1, 2, ..., NO, Figure 1 represents the process described.

PDQ ALGORITHM

The successive division of subsets into equal subsets, as illustrated, will cause the 1 element to migrate to the proper position in the array; but in the worst case, the 2 element will migrate toward the middle position of the array. The 2 element can be forced left by dividing the subsets unequally, as shown in Figure 2. Let the kth interval of comparison be defined as the integer part of I_k = NO * F_k, where 0 < F < 1. A code to achieve the partitioning is given in Listing 1. F = .8 seems to produce a complete sort on a uniform random distribution; however, the author has not been able to characterize the optimal value for F for a particular given distribution. Observation of empirical test results suggests that certain structured distributions are difficult to sort; therefore, the PDQ algorithm may be useful as a test of randomness.

SORT CODE TESTING ALGORITHM

DISTRIBUTIONS

The sort code was tested by a control program (Listings 2A to 2D) that generates strings of a specified length from six different distributions:

1. Sorted arrays: the elements of the sequence 1, 2, 3, ..., NO.

2. Arrays in reverse order: the elements of the sequence NO, NO-1, ..., 2, 1.

3. Random arrays: numbers from a uniform distribution over the interval (0, 1.). The random numbers are multiplied by 100,000 to minimize duplicate numbers when converted to integers.
4. Arrays almost in sort: the sequence, 1, 2, ..., NO with a specified number of elements, chosen at random, set to values taken from a uniform random distribution.

5. Arrays of equal length sorted blocks: this is the distribution described in (3) sorted in successive segments of a specified length.

6. Constant value arrays: a sequence of numbers of equal value. These distributions were selected with a view to test algorithm behavior on distributions that might be encountered in practical applications, such as (3) and (4), or to demonstrate unusual characteristics.

Elapsed time to sort is measured, and a count of departures from a monotonic sequence is made. If this count is greater than 0, an error message is printed.

TESTING ENVIRONMENT

All test runs were conducted on a CDC CYBER 74, with the same level of optimization (OPT = 2). Observations of repeated runs on the CYBER 74, operating in a time-sharing mode, suggest that time measurements can vary about 20%; however, by specifying large arrays to sort (90K), the job will be made to run on the machine in a more dedicated configuration. In this dedicated mode, elapsed times are highly reproducible. Sort times are sensitive to the values taken for F, as shown in Figure 3.

Machine Dependancy

An apparent anomaly is that it takes more time to sort a sorted string than to sort a string that has an initial uniform random distribution. This seems to indicate that it takes more time to execute a branch instruction than the three arithmetic replacement statements involved in the number exchange. This is true in the aggregate for the repetitive execution of the code in the DO loop used to compare and exchange the elements of the NUM array.

The CYBER 74 is a stack machine. Up to seven words of packed instructions containing up to 28 instructions can be retained in registers constituting an instruction stack. This device can increase instruction execution speed by reducing memory references; however, a forward branch in the stack “voids the stack”, therefore is an expensive operation. For this reason, PDQ will sort faster if the “less than equal” test is replaced by a “less than” test in the exchange algorithm. This increased speed is demonstrated in the decreased times required to sort a constant value array, as compared to the time required to sort an array already in sort.

Markedly different results may be expected on some other computer systems. The expected time relationship may be obtained by replacing the DO loop with an IF loop, where the branch will be in a backward direction in the instruction stack (Listing 3). Unfortunately, code optimization is not as intensive now, and overall sort times are increased.
Figure 1. Successive Partition of Sets into Equal Subsets.

PASS 1
\[ l_1 = \text{NO}/2 \]

PASS 2
\[ l_2 = l_1/2 \]

\[ \vdots \]

PASS n
\[ l_n = 1 \]

Figure 2. PDQ Partitioning Process.

PASS 1
\[ l_1 = \text{NO}/2 \]

PASS 2
\[ l_2 = \text{NO}/3 \]

PASS 3
\[ l_3 = \text{NO}/4 \]

\[ \vdots \]

PASS n
\[ l_n = 1 \]
SUBROUTINE SORT(NUM,N0)
DIMENSION N0(100)
C
10 A = NO
   N = A + 1
   IF (I .LE. 0) GO TO 20
   RETURN
   K = NO - I
   DO 15 J = 1, K
      IF (NUM(J) .LE. NUM(I+J)) GO TO 15
      MAX = NUM(J)
      NUM(J) = NUM(I+J)
      NUM(I+J) = MAX
   15 CONTINUE
   GO TO 10
C
END

PROGRAM SORT(INPUT,OUTPUT,TAPES=INPUT,TAPE6=OUTPUT,PUNCH)
COMMON TIME(71),NO,A1960601
DATA ICEED/Z6/
READ (5,100) IP1,IP2,IP3,ALF,N
WRITE(6,111) IP1,IP2,IP3,ALF,N
GO TO 10 N = 1,NO
10 CALL TEST(1,48M)
   LISTING 1
   SORTED ARRAYS)
   DO 20 N = 1, NO
      CALL TEST(2,48M)
         ARRAYS SORTED IN REVERSE ORDER
   20 CALL RANSET(ICEED)
   DO 30 N = 1, NO
      CALL TEST(3,48M)
         RANDOM ARRAYS
   30 A(N) = 1,NO
   DO 45 I = 1, M
      N = RANF(8) = 100000.
      A(N) = A(N+1,NO)
   45 CONTINUE
   CALL TEST(4,48M)
   CALL RANSET(ICEED)
   DO 50 N = 1, NO
      CALL STRING(M,48M)
         ARRAYS OF EQUAL LENGTH SORTED BLOCKS
   50 A(N) = RANF(8) = 100000.
   CALL TEST(5,48M)
   CALL RANSET(ICEED)
   DO 60 N = 1, NO
      CALL TEST(6,48M)
         CONSTANT VALUE ARRAYS
   60 CONTINUE
   FORMAT(3110,A10,4110)
   CONTINUE
   FORMAT(1H1,3110,2X,A10,4110)
C
END

LISTING 2A
SUBROUTINE TEST(AB:160000)
DIMENSION ALF(1), NUM (100000)
EQUIVALENCE (A(1), NUM(1))
CALL TIME(K,K)
CALL SORT(K,N)
CALL TIME(K,K)
CALL CHECK
WRITE(6,200) (ALF(I).I=1,4),NO,TIME(K)
N1 = NO = 0
CALL OUT(N1,NO)
RETURN
C
FORMAT(1H '4A10.I6,F10.2')
END

C
SUBROUTINE TIME(NOD:AN36000)
TP = SECOND(8)
ENTRY ETIME:
TC = SECOND(8)
TIME(N1) = TC - TP
TP = TC
ENTRY CHECK
IE = 0
DO 20 J = 2, NO
IF(I(J-1) .GT. A(J)) IC = IE + 1
WRITE(6,100) IE
ENTRY OUT
WRITE(6,300) (A(N).N=N1.N2)
FORMAT(1H '35X 12ERROR COUNT=',I6)
FORMAT(1H '65X 9F7.6')
C
END

C
SUBROUTINE TIMING(N196000)
DO 30 N1 = 1, NO, M
N2 = NUM(N1+M-1,NO)
I = 1
IF(I .LE. 1) GO TO 36
DO 25 J = 1, N2
IF(I .LT. N2 - I) I = 1
GO TO 36
10 IF(I .LE. 1) GO TO 25
IF(A(I) .LE. A(I+L)) GO TO 25
BIG = A(I)
BIG = A(I+L)
L = 1
IF(IE .GE. N1) GO TO 15
CONTINUE
GO TO 10
CONTINUE
RETURN
C
END
<table>
<thead>
<tr>
<th>F</th>
<th>PASSES</th>
<th>TIME1</th>
<th>TIME2</th>
<th>ERR1</th>
<th>ERR2</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>0</td>
<td>26</td>
<td>120</td>
<td>5830</td>
<td>5809</td>
</tr>
<tr>
<td>25</td>
<td>0</td>
<td>27</td>
<td>127</td>
<td>4939</td>
<td>4914</td>
</tr>
<tr>
<td>30</td>
<td>0</td>
<td>28</td>
<td>131</td>
<td>5046</td>
<td>4991</td>
</tr>
<tr>
<td>35</td>
<td>0</td>
<td>29</td>
<td>139</td>
<td>5016</td>
<td>4984</td>
</tr>
<tr>
<td>40</td>
<td>0</td>
<td>30</td>
<td>149</td>
<td>4973</td>
<td>4967</td>
</tr>
<tr>
<td>45</td>
<td>1</td>
<td>31</td>
<td>151</td>
<td>5005</td>
<td>4956</td>
</tr>
<tr>
<td>50</td>
<td>1</td>
<td>32</td>
<td>157</td>
<td>4971</td>
<td>4921</td>
</tr>
<tr>
<td>55</td>
<td>1</td>
<td>33</td>
<td>162</td>
<td>4964</td>
<td>4900</td>
</tr>
<tr>
<td>60</td>
<td>1</td>
<td>34</td>
<td>164</td>
<td>4934</td>
<td>4864</td>
</tr>
<tr>
<td>65</td>
<td>1</td>
<td>35</td>
<td>166</td>
<td>4912</td>
<td>4844</td>
</tr>
<tr>
<td>70</td>
<td>1</td>
<td>36</td>
<td>169</td>
<td>4908</td>
<td>4824</td>
</tr>
<tr>
<td>75</td>
<td>2</td>
<td>37</td>
<td>170</td>
<td>4894</td>
<td>4800</td>
</tr>
<tr>
<td>80</td>
<td>2</td>
<td>38</td>
<td>171</td>
<td>4880</td>
<td>4774</td>
</tr>
<tr>
<td>85</td>
<td>2</td>
<td>39</td>
<td>172</td>
<td>4869</td>
<td>4750</td>
</tr>
<tr>
<td>90</td>
<td>2</td>
<td>40</td>
<td>174</td>
<td>4859</td>
<td>4724</td>
</tr>
<tr>
<td>95</td>
<td>2</td>
<td>41</td>
<td>175</td>
<td>4849</td>
<td>4698</td>
</tr>
<tr>
<td>100</td>
<td>3</td>
<td>42</td>
<td>176</td>
<td>4838</td>
<td>4670</td>
</tr>
<tr>
<td>105</td>
<td>3</td>
<td>43</td>
<td>177</td>
<td>4828</td>
<td>4640</td>
</tr>
<tr>
<td>110</td>
<td>3</td>
<td>44</td>
<td>178</td>
<td>4818</td>
<td>4604</td>
</tr>
<tr>
<td>115</td>
<td>3</td>
<td>45</td>
<td>179</td>
<td>4808</td>
<td>4568</td>
</tr>
<tr>
<td>120</td>
<td>3</td>
<td>46</td>
<td>180</td>
<td>4798</td>
<td>4530</td>
</tr>
<tr>
<td>125</td>
<td>3</td>
<td>47</td>
<td>181</td>
<td>4788</td>
<td>4490</td>
</tr>
<tr>
<td>130</td>
<td>3</td>
<td>48</td>
<td>182</td>
<td>4777</td>
<td>4448</td>
</tr>
<tr>
<td>135</td>
<td>3</td>
<td>49</td>
<td>183</td>
<td>4767</td>
<td>4408</td>
</tr>
<tr>
<td>140</td>
<td>3</td>
<td>50</td>
<td>184</td>
<td>4756</td>
<td>4366</td>
</tr>
<tr>
<td>145</td>
<td>3</td>
<td>51</td>
<td>185</td>
<td>4746</td>
<td>4320</td>
</tr>
<tr>
<td>150</td>
<td>3</td>
<td>52</td>
<td>186</td>
<td>4736</td>
<td>4278</td>
</tr>
<tr>
<td>155</td>
<td>3</td>
<td>53</td>
<td>187</td>
<td>4726</td>
<td>4232</td>
</tr>
<tr>
<td>160</td>
<td>3</td>
<td>54</td>
<td>188</td>
<td>4716</td>
<td>4184</td>
</tr>
<tr>
<td>165</td>
<td>3</td>
<td>55</td>
<td>189</td>
<td>4706</td>
<td>4132</td>
</tr>
<tr>
<td>170</td>
<td>3</td>
<td>56</td>
<td>190</td>
<td>4696</td>
<td>4078</td>
</tr>
<tr>
<td>175</td>
<td>3</td>
<td>57</td>
<td>191</td>
<td>4684</td>
<td>4020</td>
</tr>
<tr>
<td>180</td>
<td>3</td>
<td>58</td>
<td>192</td>
<td>4674</td>
<td>3958</td>
</tr>
<tr>
<td>185</td>
<td>3</td>
<td>59</td>
<td>193</td>
<td>4663</td>
<td>3890</td>
</tr>
<tr>
<td>190</td>
<td>3</td>
<td>60</td>
<td>194</td>
<td>4652</td>
<td>3824</td>
</tr>
<tr>
<td>195</td>
<td>3</td>
<td>61</td>
<td>195</td>
<td>4642</td>
<td>3750</td>
</tr>
<tr>
<td>200</td>
<td>3</td>
<td>62</td>
<td>196</td>
<td>4632</td>
<td>3678</td>
</tr>
<tr>
<td>205</td>
<td>3</td>
<td>63</td>
<td>197</td>
<td>4622</td>
<td>3600</td>
</tr>
<tr>
<td>210</td>
<td>3</td>
<td>64</td>
<td>198</td>
<td>4612</td>
<td>3524</td>
</tr>
<tr>
<td>215</td>
<td>3</td>
<td>65</td>
<td>199</td>
<td>4602</td>
<td>3450</td>
</tr>
<tr>
<td>220</td>
<td>3</td>
<td>66</td>
<td>200</td>
<td>4592</td>
<td>3377</td>
</tr>
<tr>
<td>225</td>
<td>3</td>
<td>67</td>
<td>201</td>
<td>4582</td>
<td>3305</td>
</tr>
<tr>
<td>230</td>
<td>3</td>
<td>68</td>
<td>202</td>
<td>4572</td>
<td>3234</td>
</tr>
<tr>
<td>235</td>
<td>3</td>
<td>69</td>
<td>203</td>
<td>4562</td>
<td>3164</td>
</tr>
<tr>
<td>240</td>
<td>3</td>
<td>70</td>
<td>204</td>
<td>4552</td>
<td>3093</td>
</tr>
<tr>
<td>245</td>
<td>3</td>
<td>71</td>
<td>205</td>
<td>4542</td>
<td>3023</td>
</tr>
<tr>
<td>250</td>
<td>3</td>
<td>72</td>
<td>206</td>
<td>4532</td>
<td>2953</td>
</tr>
<tr>
<td>255</td>
<td>3</td>
<td>73</td>
<td>207</td>
<td>4522</td>
<td>2883</td>
</tr>
<tr>
<td>260</td>
<td>3</td>
<td>74</td>
<td>208</td>
<td>4512</td>
<td>2813</td>
</tr>
<tr>
<td>265</td>
<td>3</td>
<td>75</td>
<td>209</td>
<td>4502</td>
<td>2743</td>
</tr>
<tr>
<td>270</td>
<td>3</td>
<td>76</td>
<td>210</td>
<td>4492</td>
<td>2674</td>
</tr>
<tr>
<td>275</td>
<td>3</td>
<td>77</td>
<td>211</td>
<td>4482</td>
<td>2604</td>
</tr>
<tr>
<td>280</td>
<td>3</td>
<td>78</td>
<td>212</td>
<td>4472</td>
<td>2534</td>
</tr>
<tr>
<td>285</td>
<td>3</td>
<td>79</td>
<td>213</td>
<td>4462</td>
<td>2464</td>
</tr>
<tr>
<td>290</td>
<td>3</td>
<td>80</td>
<td>214</td>
<td>4452</td>
<td>2394</td>
</tr>
<tr>
<td>295</td>
<td>3</td>
<td>81</td>
<td>215</td>
<td>4442</td>
<td>2324</td>
</tr>
<tr>
<td>300</td>
<td>3</td>
<td>82</td>
<td>216</td>
<td>4432</td>
<td>2254</td>
</tr>
<tr>
<td>305</td>
<td>3</td>
<td>83</td>
<td>217</td>
<td>4422</td>
<td>2184</td>
</tr>
<tr>
<td>310</td>
<td>3</td>
<td>84</td>
<td>218</td>
<td>4412</td>
<td>2114</td>
</tr>
<tr>
<td>315</td>
<td>3</td>
<td>85</td>
<td>219</td>
<td>4402</td>
<td>2044</td>
</tr>
<tr>
<td>320</td>
<td>3</td>
<td>86</td>
<td>220</td>
<td>4392</td>
<td>1974</td>
</tr>
<tr>
<td>325</td>
<td>3</td>
<td>87</td>
<td>221</td>
<td>4382</td>
<td>1904</td>
</tr>
<tr>
<td>330</td>
<td>3</td>
<td>88</td>
<td>222</td>
<td>4372</td>
<td>1834</td>
</tr>
<tr>
<td>335</td>
<td>3</td>
<td>89</td>
<td>223</td>
<td>4362</td>
<td>1764</td>
</tr>
<tr>
<td>340</td>
<td>3</td>
<td>90</td>
<td>224</td>
<td>4352</td>
<td>1694</td>
</tr>
<tr>
<td>345</td>
<td>3</td>
<td>91</td>
<td>225</td>
<td>4342</td>
<td>1624</td>
</tr>
<tr>
<td>350</td>
<td>3</td>
<td>92</td>
<td>226</td>
<td>4332</td>
<td>1554</td>
</tr>
<tr>
<td>355</td>
<td>3</td>
<td>93</td>
<td>227</td>
<td>4322</td>
<td>1484</td>
</tr>
<tr>
<td>360</td>
<td>3</td>
<td>94</td>
<td>228</td>
<td>4312</td>
<td>1414</td>
</tr>
<tr>
<td>365</td>
<td>3</td>
<td>95</td>
<td>229</td>
<td>4302</td>
<td>1344</td>
</tr>
<tr>
<td>370</td>
<td>3</td>
<td>96</td>
<td>230</td>
<td>4292</td>
<td>1274</td>
</tr>
<tr>
<td>375</td>
<td>3</td>
<td>97</td>
<td>231</td>
<td>4282</td>
<td>1204</td>
</tr>
<tr>
<td>380</td>
<td>3</td>
<td>98</td>
<td>232</td>
<td>4272</td>
<td>1134</td>
</tr>
<tr>
<td>385</td>
<td>3</td>
<td>99</td>
<td>233</td>
<td>4262</td>
<td>1064</td>
</tr>
<tr>
<td>390</td>
<td>3</td>
<td>100</td>
<td>234</td>
<td>4252</td>
<td>994</td>
</tr>
</tbody>
</table>

Figure 3. Elapsed Time as a Function of F for the PDQ Partial Sort (Listing 1).
VARIATIONS

The simplicity of the PDQ code lends it to numerous variations; for instance, PDQ modules can be stacked with different values for F in each module. Changing the direction of the comparisons on alternate passes causes sorting to occur in fewer passes but at the cost of increased complexity (Listing 4). Listing 5 is a format of PDQ using a partitioning scheme involving the logarithm to the base 12.

SUBROUTINE SORT(F)
COMMON NUM(10000),NO

C PDQ IF LOOP
A = NO
10 A = A+F
    IF(I+LE 0) RETURN
    K = NO - I
    DO 15 J = 1,K
        IF(NUM(J).LE.NUM(I+J)) GO TO 15
        MAX = NUM(J)
        NUM(J) = NUM(I+J)
        NUM(I+J) = MAX
    CONTINUE
15 CONTINUE
    GO TO 10
C END

LISTING 3

SUBROUTINE SORT(F)
COMMON NUM(10000),NO

C PDQ ALTERNATE
A = NO
10 A = A+F
    IF(I+LE 0) RETURN
    K = NO - I
    DO 15 J = 1,K
        IF(NUM(J).LE.NUM(I+J)) GO TO 15
        MAX = NUM(J)
        NUM(J) = NUM(I+J)
        NUM(I+J) = MAX
    CONTINUE
15 CONTINUE
    GO TO 10
C END

LISTING 4
The original intention of the sort strategy is achieved, with the assurance of a sort by following PDQ with a sort by direct insertion. The insertion code is derived from PDQ by the addition of a few lines of code (Listing 6).

Insertion means the addition of numbers to an existing sorted string (which initially may be of length 1) by inserting the new element into, or at the ends of the sorted string to form a new sorted string of increased length. For any distribution, the sort is completed in one pass. An INSERTION sort is to be distinguished from a BUBBLE sort where successive elements are selected and added to one end of a string, initially of length 0. The BUBBLE sort may require up to NO-I passes to complete the sort. Tests show the INSERTION sort to be more efficient than the BUBBLE sort (Listing 7).

If the features of the PDQ sort and the INSERTION sort are combined, a particularly efficient algorithm resembling the SHELL sort is obtained (Listing 8). The specification of an optimal sequence of intervals to control partitioning is a difficult task; however, if a geometric sequence is assumed, an F can empirically be found which will yield minimum sort times for a given distribution (Figure 4). If two or more values of F produce minimal elapsed times, the smallest of these values should be selected to reduce sort times on sorted or nearly sorted strings.

The SHELL sort seems to have been intended as a type of merge algorithm. The term "merge" may be used in several ways:

1. The combination of two or more sorted strings into a resultant sorted string irrespective of any particular algorithm.

2. A specific method by which sorted strings can be combined efficiently into resultant sorted strings.

---

SUBROUTINE SORT(NUM,NO)
DIMENSION NUM(NO)

10 A = NO
A = A+7
IF(I .LE. 6) GO TO 2
IF(MOD(I,2) .EQ. 1) I = I + 1
K = NO - 1
DO 15 J = 1,K
IF(NUM(J) .LE. NUM(I+J)) GO TO 15
MAX = NUM(J)
NUM(J) = NUM(I+J)
NUM(I+J) = MAX
15 CONTINUE
GO TO 10

20 K = NO - 1
GO 30 J = 1,K
IF(NUM(I) .LE. NUM(J+1)) GO TO 30
MAX = NUM(I)
NUM(I) = NUM(J+1)
NUM(J+1) = MAX
I = J
30 CONTINUE
GO TO 25
RETURN

LISTING 6

C
SUBROUTINE SORT(IF)
COMMON NUM(10000),NO

5 K = NO - 1
L = 1
DO 14 J = 1,K
IF(NUM(J) .LE. NUM(J+1)) GO TO 10
MAX = NUM(J)
NUM(J) = NUM(J+1)
NUM(J+1) = MAX
10 CONTINUE
IF(L .EQ. 1) RETURN
K = L - 1
GO TO 5
END

LISTING 7

C
SUBROUTINE SORT(IF)
COMMON NUM(10000),NO,TIME(7)

10 IF(I .LE. 1) RETURN
A = AF
T = I
IF(I .LT. 1) I = 1
K = NO - 1
DO 20 J = 1,K
IF(NUM(I) .LE. NUM(I+L)) GO TO 20
MAX = NUM(I)
NUM(I) = NUM(I+L)
NUM(I+L) = MAX
20 CONTINUE
GO TO 10
END

LISTING 8
<table>
<thead>
<tr>
<th>NO</th>
<th>F</th>
<th>TIME IN SECONDS</th>
<th>NO</th>
<th>F</th>
<th>TIME IN SECONDS</th>
<th>NO</th>
<th>F</th>
<th>TIME IN SECONDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>268</td>
<td>2.43</td>
<td>10000</td>
<td>285</td>
<td>2.33</td>
<td>10000</td>
<td>295</td>
<td>2.25</td>
</tr>
<tr>
<td>10000</td>
<td>269</td>
<td>2.33</td>
<td>10000</td>
<td>296</td>
<td>2.22</td>
<td>10000</td>
<td>305</td>
<td>2.15</td>
</tr>
<tr>
<td>10000</td>
<td>270</td>
<td>2.21</td>
<td>10000</td>
<td>297</td>
<td>2.21</td>
<td>10000</td>
<td>306</td>
<td>2.15</td>
</tr>
<tr>
<td>10000</td>
<td>271</td>
<td>2.13</td>
<td>10000</td>
<td>298</td>
<td>2.23</td>
<td>10000</td>
<td>307</td>
<td>2.10</td>
</tr>
<tr>
<td>10000</td>
<td>272</td>
<td>2.14</td>
<td>10000</td>
<td>299</td>
<td>2.20</td>
<td>10000</td>
<td>308</td>
<td>2.10</td>
</tr>
<tr>
<td>10000</td>
<td>273</td>
<td>2.05</td>
<td>10000</td>
<td>300</td>
<td>2.09</td>
<td>10000</td>
<td>309</td>
<td>2.05</td>
</tr>
<tr>
<td>10000</td>
<td>274</td>
<td>2.09</td>
<td>10000</td>
<td>301</td>
<td>2.09</td>
<td>10000</td>
<td>310</td>
<td>2.09</td>
</tr>
<tr>
<td>10000</td>
<td>275</td>
<td>2.09</td>
<td>10000</td>
<td>302</td>
<td>2.09</td>
<td>10000</td>
<td>311</td>
<td>2.09</td>
</tr>
<tr>
<td>10000</td>
<td>276</td>
<td>2.09</td>
<td>10000</td>
<td>303</td>
<td>2.09</td>
<td>10000</td>
<td>312</td>
<td>2.09</td>
</tr>
<tr>
<td>10000</td>
<td>277</td>
<td>2.09</td>
<td>10000</td>
<td>304</td>
<td>2.09</td>
<td>10000</td>
<td>313</td>
<td>2.09</td>
</tr>
<tr>
<td>10000</td>
<td>278</td>
<td>2.09</td>
<td>10000</td>
<td>305</td>
<td>2.09</td>
<td>10000</td>
<td>314</td>
<td>2.09</td>
</tr>
<tr>
<td>10000</td>
<td>279</td>
<td>2.09</td>
<td>10000</td>
<td>306</td>
<td>2.09</td>
<td>10000</td>
<td>315</td>
<td>2.09</td>
</tr>
<tr>
<td>10000</td>
<td>280</td>
<td>2.09</td>
<td>10000</td>
<td>307</td>
<td>2.09</td>
<td>10000</td>
<td>316</td>
<td>2.09</td>
</tr>
<tr>
<td>10000</td>
<td>281</td>
<td>2.09</td>
<td>10000</td>
<td>308</td>
<td>2.09</td>
<td>10000</td>
<td>317</td>
<td>2.09</td>
</tr>
<tr>
<td>10000</td>
<td>282</td>
<td>2.09</td>
<td>10000</td>
<td>309</td>
<td>2.09</td>
<td>10000</td>
<td>318</td>
<td>2.09</td>
</tr>
<tr>
<td>10000</td>
<td>283</td>
<td>2.09</td>
<td>10000</td>
<td>310</td>
<td>2.09</td>
<td>10000</td>
<td>319</td>
<td>2.09</td>
</tr>
<tr>
<td>10000</td>
<td>284</td>
<td>2.09</td>
<td>10000</td>
<td>311</td>
<td>2.09</td>
<td>10000</td>
<td>320</td>
<td>2.09</td>
</tr>
<tr>
<td>10000</td>
<td>285</td>
<td>2.09</td>
<td>10000</td>
<td>312</td>
<td>2.09</td>
<td>10000</td>
<td>321</td>
<td>2.09</td>
</tr>
<tr>
<td>10000</td>
<td>286</td>
<td>2.09</td>
<td>10000</td>
<td>313</td>
<td>2.09</td>
<td>10000</td>
<td>322</td>
<td>2.09</td>
</tr>
<tr>
<td>10000</td>
<td>287</td>
<td>2.09</td>
<td>10000</td>
<td>314</td>
<td>2.09</td>
<td>10000</td>
<td>323</td>
<td>2.09</td>
</tr>
<tr>
<td>10000</td>
<td>288</td>
<td>2.09</td>
<td>10000</td>
<td>315</td>
<td>2.09</td>
<td>10000</td>
<td>324</td>
<td>2.09</td>
</tr>
<tr>
<td>10000</td>
<td>289</td>
<td>2.09</td>
<td>10000</td>
<td>316</td>
<td>2.09</td>
<td>10000</td>
<td>325</td>
<td>2.09</td>
</tr>
<tr>
<td>10000</td>
<td>290</td>
<td>2.09</td>
<td>10000</td>
<td>317</td>
<td>2.09</td>
<td>10000</td>
<td>326</td>
<td>2.09</td>
</tr>
<tr>
<td>10000</td>
<td>291</td>
<td>2.09</td>
<td>10000</td>
<td>318</td>
<td>2.09</td>
<td>10000</td>
<td>327</td>
<td>2.09</td>
</tr>
<tr>
<td>10000</td>
<td>292</td>
<td>2.09</td>
<td>10000</td>
<td>319</td>
<td>2.09</td>
<td>10000</td>
<td>328</td>
<td>2.09</td>
</tr>
<tr>
<td>10000</td>
<td>293</td>
<td>2.09</td>
<td>10000</td>
<td>320</td>
<td>2.09</td>
<td>10000</td>
<td>329</td>
<td>2.09</td>
</tr>
<tr>
<td>10000</td>
<td>294</td>
<td>2.09</td>
<td>10000</td>
<td>321</td>
<td>2.09</td>
<td>10000</td>
<td>330</td>
<td>2.09</td>
</tr>
<tr>
<td>10000</td>
<td>295</td>
<td>2.09</td>
<td>10000</td>
<td>322</td>
<td>2.09</td>
<td>10000</td>
<td>331</td>
<td>2.09</td>
</tr>
<tr>
<td>10000</td>
<td>296</td>
<td>2.09</td>
<td>10000</td>
<td>323</td>
<td>2.09</td>
<td>10000</td>
<td>332</td>
<td>2.09</td>
</tr>
<tr>
<td>10000</td>
<td>297</td>
<td>2.09</td>
<td>10000</td>
<td>324</td>
<td>2.09</td>
<td>10000</td>
<td>333</td>
<td>2.09</td>
</tr>
<tr>
<td>10000</td>
<td>298</td>
<td>2.09</td>
<td>10000</td>
<td>325</td>
<td>2.09</td>
<td>10000</td>
<td>334</td>
<td>2.09</td>
</tr>
<tr>
<td>10000</td>
<td>299</td>
<td>2.09</td>
<td>10000</td>
<td>326</td>
<td>2.09</td>
<td>10000</td>
<td>335</td>
<td>2.09</td>
</tr>
<tr>
<td>10000</td>
<td>300</td>
<td>2.09</td>
<td>10000</td>
<td>327</td>
<td>2.09</td>
<td>10000</td>
<td>336</td>
<td>2.09</td>
</tr>
</tbody>
</table>

Figure 4. Elapsed Time as a Function of F for Distribution 1 (Listing 8).
Generally, the SHELL sort does not qualify as a merge algorithm, but it can be made to operate as such under definition (1) by selecting \( F = 0.5 \) and the first interval as the largest power of 2 less than \( N \).

This choice of \( F \) turns out to be one of the worst possible. To see why, it may be constructive to consider this procedure as a type of distribution sort. Consider the sequence of 16 numbers, 1, 2, \ldots, 16 where the integers represent the position in an array of numbers to be sorted. Take the first interval of comparison, \( I_1 \) equal to 8 and \( F = 0.5 \), then the following illustration can be used:

\[
\begin{array}{cccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 \\
\end{array}
\]

Pass 1

\( I_1 = 8 \)

To suggest that the array to be sorted has been divided into eight subsequences represented by the columns. The numbers in each column are to be sorted in ascending order from top to bottom; i.e., the number in position 1 is to be less than or equal to the number in position 9, etc.

The following illustration represents the next pass:

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
5 & 6 & 7 & 8 \\
9 & 10 & 11 & 12 \\
13 & 14 & 15 & 16 \\
\end{array}
\]

Pass 2

\( I_2 = 4 \)

Note that: (1) the subsequences are reduced in number in proportion to \( F \), (2) the length of the subsequences is increased in inverse proportion to \( F \), and (3) the elements of a given column are composed alternately of the \( n \) and \( n + I_1/2 \) columns of the previous pass, \( n = 1, 2, 3, I_1/2 \). Consequently, there is a high degree of order in any column and an element is likely to be close to its sorted position in a subsequence, and an INSERTION sort or BUBBLE sort applied to a column may be expected to operate faster than on a random distribution of the same length. The worst case in this example occurs when the even and odd columns represent disjoint ranges, and the set of numbers from the even columns contains the smaller numbers. Inssofar as the median of each column represents the median of the entire distribution, subsequent passes may be expected to produce increasingly well-ordered subsequences representative of the entire sequence; therefore, needing a minimum number of comparisons and exchanges to sort. However, certain difficulties can arise.

In the following example, the array notation of the previous illustration will be retained, but now the integers represent the actual numbers in the array that are to be sorted. Given the sequence 3, 5, 6, 7, 2, 8, 9, 10, 4, 11, 12, 13, 1, 14, 15, 16, let \( F = 0.5 \) and \( I_1 = 8 \), we have:

\[
\begin{array}{cccccccc}
3 & 5 & 6 & 7 & 2 & 8 & 9 & 10 \\
4 & 11 & 12 & 13 & 1 & 14 & 15 & 16 \\
\end{array}
\]

Pass 1

\( I_1 = 8 \)
All columns but the fifth are in sort; therefore, the 1 and 2 elements are exchanged. This could be called an unfavorable exchange, as it improves the order of the array very little; both the 1 and the 2 should be in the first row. After the exchange, pass 2 can be represented:

<table>
<thead>
<tr>
<th>3</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>( I_2 = 4 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>( I_2 = 4 )</td>
</tr>
<tr>
<td>4</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>( I_2 = 4 )</td>
</tr>
<tr>
<td>2</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>( I_2 = 4 )</td>
</tr>
</tbody>
</table>

The columns are well-ordered except column 1. After the sort we have:

<table>
<thead>
<tr>
<th>1</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td>4</td>
<td>14</td>
<td>15</td>
<td>16</td>
</tr>
</tbody>
</table>

Column 1 has a median atypical of the array, and most of its elements are far from their final position. It will take a large number of small steps to move them into position in later passes. The elements in column 1 after row 1 may be thought of as blocking elements because they inhibit efficient distribution of the array elements in the early stages of the sort when an element can proceed toward its destination by large steps. To follow the example further:

<table>
<thead>
<tr>
<th>1</th>
<th>5</th>
<th>( I_3 = 2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>6</td>
</tr>
<tr>
<td>12</td>
<td>13</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>14</td>
<td>12</td>
</tr>
<tr>
<td>15</td>
<td>16</td>
<td>15</td>
</tr>
</tbody>
</table>

The last pass with \( I_4 = 1 \) will require a large number of exchanges to complete the sort. The collection of blocking elements can be prevented by a number of methods, such as sorting up a diagonal from left to right, following a column sort. But most of the comparisons will not result in an exchange due to a high degree of order produced by the previous pass,
and those exchanges that do result are bought at the expense of an extra pass. Similar objections may be raised to other modifications to the algorithm to eliminate the blocking elements. Examination of the problem reveals that the accumulation of blocking elements from pass to pass is particularly severe if the rows in the array are divided into an integral number of rows in the next pass; therefore, .25 and .5 are bad values for F. This can be verified readily by rewriting the sort algorithm to include counts of primary exchanges and secondary exchanges on a pass-by-pass basis, and printing these values (as well as their cumulative totals), for selected values of F (Figures 5 and 6). It is interesting to note that the minimum sort times do not correspond to the values of F that result in minimum exchanges, because as F becomes larger, there is an increase in overhead associated with the increase in the number of passes. The elapsed times shown are distorted by the code modification required for the accumulation and printing of the statistics.

A study of the exchange counts suggests that the blocking phenomena come into play whenever there is a series of intervals that have factors in common, and that the problem becomes more severe as F decreases; therefore, the intervals should be relatively prime. Consider the first column of the array on the mth pass. The location of a blocking element in the first column can be expressed as $I + M*P_m$. On the next pass, if that same element is to appear in the same column, it must have a location expressible as $I + N*P_n$, where $P_n < P_m$. Then $N*P_n = M*P_m$, and $N = P_m$, $M = P_n$ if $P_m$, $P_n$ are relatively prime. But these values for M and N are impossible during the earlier passes of the sort for large arrays and the range of F considered here. To test this hypothesis, sequences of relatively prime numbers corresponding to the geometric sequences associated to values for F ($F = .2$ to $F = .46$) were used in the sort algorithm (Listing 9) to conduct elapsed time tests (Figure 7). The tests seem to confirm the hypothesis; however, the use of prime sequences is only a partial solution. A blocking element may be replaced in a column with another element that also serves as a blocking element. This situation is likely to occur when sorting arrays of equal length sorted blocks. Counts of exchanges and comparisons for the prime sequences are given in Figure 8.

**PRIME SORTING ALGORITHM**

The prime sequence corresponding to $F = .3$ was selected with a view to minimum combined elapsed times for both random and sorted distributions for the algorithm PRIME SORT (Listing 10). This algorithm also incorporates a change that significantly improves the efficiency of exchanges, especially secondary exchanges.

The improvement in execution speed was largely lost when the data was passed through the CALL list. This illustrates a compilation problem that affects the various algorithms given here to differing extents; the PDQ codes were degraded least by passing data through the CALL list. Efficient compilation is important because the PDQ, PRIME, and DISTRIBUTION sort codes given here derive their speed from compact code that requires a minimum of instructions to execute, and from improved partitioning schemes. Full realization of the potential of the code requires effective register assignment to indices, etc., but variables passed through CALL lists inhibit optimal compilation. Generally, an improvement in performance may be expected when data is passed to a subroutine through COMMON storage (Figure 9).
Figure 5. Counts of Exchanges and Comparisons as a Function of $F$ for the Distribution #1 Algorithm.
Figure 6. Cumulative Counts of Exchanges and Comparisons as a Function of T for the Distribution Algorithm.
SUBROUTINE SORT(NUM,N0,N1)
DIMENSION NUM(N0),INK(16,27)

DATA(INK(I,J),I=1,16, J=1,19)
*99625,7,89
*264390,7,9
*197965,7,9
*117598,7,9
*00847,7,9
*65537,262144,6,9
*47801,184179,6,9
*35607,123137,6,9
*75679,9637,17613,5,9
*19991,68827,237985,5,9
*119741,504921,166935,5,9
*117311,37331,122007,5,9
*9091,28499,85071,77556,4,9
*71099,21557,65293,197834,4,9
*7591,16477,48437,147473,4,9
*4441,12589,36251,107973,4,9
*5547,9851,27761,75979,4,9
*2251,76399,20789,56207,151908,3,9
*2297,6605,15923,41911,110395,3,9

DATA(INK(I,J),I=1,16, J=20,27)/1,5,23,127,631,1312,15629,78121, F = 20
*99625,7,89
*264390,7,9
*197965,7,9
*117598,7,9
*00847,7,9
*65537,262144,6,9
*47801,184179,6,9
*35607,123137,6,9
*75679,9637,17613,5,9
*19991,68827,237985,5,9
*119741,504921,166935,5,9
*117311,37331,122007,5,9
*9091,28499,85071,77556,4,9
*71099,21557,65293,197834,4,9
*7591,16477,48437,147473,4,9
*4441,12589,36251,107973,4,9
*5547,9851,27761,75979,4,9
*2251,76399,20789,56207,151908,3,9
*2297,6605,15923,41911,110395,3,9

DO 5 IN = 1,16
IF(N0 .LE. INK(IN,N)) GO TO 10
E CONTINUE

10 TM = TM - 1
IF(IN .LE. 0) RETURN
I = INK(IN,N)
K = 40 - I

DO 20 J = 1,K
I = I
4 INUM(L) .LE. NUM(I+L)) GO TO 20
MAX = NUM(L)
NUM(L) = NUM(I+L)
NUM(I+L) = MAX
L = L - I
TEL GT. 01 GO TO 1F
20 CONTINUE
GO TO 10
7 END

LISTING 9
### Figure 7. Elapsed Time for Prime Sequences, F = 2 to F = 46.

<table>
<thead>
<tr>
<th>TIME IN SECONDS</th>
<th>PRIME</th>
<th>NO</th>
<th>RANDOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

### Figure 8. Cumulative Counts of Exchanges and Comparisons for Prime Sequences, F = 2 to F = 46.

<table>
<thead>
<tr>
<th>PRIMARY EXCHANGE</th>
<th>SECONDARY EXCHANGE</th>
<th>PRIMARY COMPARISON</th>
<th>SECONDARY COMPARISON</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

---

10000
1000 PRIME L9
TIME IN SECONDS
0
Assembly language coding can minimize these problems, and candidates for that purpose are given in Listings 11 and 12. An algorithm suitable for in-line code is given in Listing 13. The effect of a prime sequence is approximated by making all intervals odd, etc.

Expected sort behavior is indicated by the expression for the number of comparisons required by the PDQ sort. The intervals of comparison are given approximately by the sequence:

\[ F^1 \times NO, F^2 \times NO, F^3 \times NO, \ldots, F^n \times NO \]

such that \( F^n \times NO = 1 \). Then \( n = -(\log NO)/(\log F) \). The number of comparisons for \( n \) passes are:

\[
K = (NO-F^1 \times NO) + (NO-F^2 \times NO) + \ldots + (NO-F^n \times NO)
\]

\[
K = n \times NO - (F + F^2 + \ldots + F^n) \times NO
\]

\[
K = \frac{\log NO}{\log F} \times NO - (F + F^2 + \ldots + F^n) \times NO
\]

This is also the expression for the primary comparisons of the DISTRIBUTION sort, but with a smaller value for \( F \).
<table>
<thead>
<tr>
<th>NO</th>
<th>D#2</th>
<th>L13</th>
<th>TIME</th>
<th>IN SECONDS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>RANDOM</td>
<td>SORTED</td>
</tr>
<tr>
<td>10000</td>
<td>200</td>
<td>.24</td>
<td>1.22</td>
<td>.33</td>
</tr>
<tr>
<td>10000</td>
<td>205</td>
<td>.22</td>
<td>1.25</td>
<td>.33</td>
</tr>
<tr>
<td>10000</td>
<td>211</td>
<td>.20</td>
<td>1.37</td>
<td>.34</td>
</tr>
<tr>
<td>10000</td>
<td>220</td>
<td>.20</td>
<td>1.21</td>
<td>.33</td>
</tr>
<tr>
<td>10000</td>
<td>225</td>
<td>.07</td>
<td>1.59</td>
<td>.33</td>
</tr>
<tr>
<td>10000</td>
<td>230</td>
<td>.11</td>
<td>1.07</td>
<td>.39</td>
</tr>
<tr>
<td>10000</td>
<td>235</td>
<td>.05</td>
<td>1.22</td>
<td>.33</td>
</tr>
<tr>
<td>10000</td>
<td>240</td>
<td>.06</td>
<td>1.07</td>
<td>.39</td>
</tr>
<tr>
<td>10000</td>
<td>245</td>
<td>.21</td>
<td>1.21</td>
<td>.36</td>
</tr>
<tr>
<td>10000</td>
<td>250</td>
<td>.06</td>
<td>1.16</td>
<td>.39</td>
</tr>
<tr>
<td>10000</td>
<td>255</td>
<td>.05</td>
<td>1.09</td>
<td>.39</td>
</tr>
<tr>
<td>10000</td>
<td>260</td>
<td>.08</td>
<td>1.22</td>
<td>.38</td>
</tr>
<tr>
<td>10000</td>
<td>265</td>
<td>.06</td>
<td>1.09</td>
<td>.38</td>
</tr>
<tr>
<td>10000</td>
<td>270</td>
<td>.06</td>
<td>1.22</td>
<td>.38</td>
</tr>
<tr>
<td>10000</td>
<td>275</td>
<td>.06</td>
<td>1.22</td>
<td>.38</td>
</tr>
<tr>
<td>10000</td>
<td>280</td>
<td>.07</td>
<td>1.09</td>
<td>.38</td>
</tr>
<tr>
<td>10000</td>
<td>285</td>
<td>.06</td>
<td>1.09</td>
<td>.38</td>
</tr>
<tr>
<td>10000</td>
<td>290</td>
<td>.06</td>
<td>1.09</td>
<td>.38</td>
</tr>
<tr>
<td>10000</td>
<td>295</td>
<td>.06</td>
<td>1.09</td>
<td>.38</td>
</tr>
<tr>
<td>10000</td>
<td>300</td>
<td>.06</td>
<td>1.09</td>
<td>.38</td>
</tr>
<tr>
<td>10000</td>
<td>305</td>
<td>.06</td>
<td>1.09</td>
<td>.38</td>
</tr>
<tr>
<td>10000</td>
<td>310</td>
<td>.06</td>
<td>1.09</td>
<td>.38</td>
</tr>
<tr>
<td>10000</td>
<td>315</td>
<td>.06</td>
<td>1.09</td>
<td>.38</td>
</tr>
<tr>
<td>10000</td>
<td>320</td>
<td>.06</td>
<td>1.09</td>
<td>.38</td>
</tr>
<tr>
<td>10000</td>
<td>325</td>
<td>.06</td>
<td>1.09</td>
<td>.38</td>
</tr>
<tr>
<td>10000</td>
<td>330</td>
<td>.06</td>
<td>1.09</td>
<td>.38</td>
</tr>
<tr>
<td>10000</td>
<td>335</td>
<td>.06</td>
<td>1.09</td>
<td>.38</td>
</tr>
<tr>
<td>10000</td>
<td>340</td>
<td>.06</td>
<td>1.09</td>
<td>.38</td>
</tr>
<tr>
<td>10000</td>
<td>345</td>
<td>.06</td>
<td>1.09</td>
<td>.38</td>
</tr>
<tr>
<td>10000</td>
<td>350</td>
<td>.06</td>
<td>1.09</td>
<td>.38</td>
</tr>
<tr>
<td>10000</td>
<td>355</td>
<td>.06</td>
<td>1.09</td>
<td>.38</td>
</tr>
<tr>
<td>10000</td>
<td>360</td>
<td>.06</td>
<td>1.09</td>
<td>.38</td>
</tr>
<tr>
<td>10000</td>
<td>365</td>
<td>.06</td>
<td>1.09</td>
<td>.38</td>
</tr>
<tr>
<td>10000</td>
<td>370</td>
<td>.06</td>
<td>1.09</td>
<td>.38</td>
</tr>
<tr>
<td>10000</td>
<td>375</td>
<td>.06</td>
<td>1.09</td>
<td>.38</td>
</tr>
<tr>
<td>10000</td>
<td>380</td>
<td>.06</td>
<td>1.09</td>
<td>.38</td>
</tr>
<tr>
<td>10000</td>
<td>385</td>
<td>.06</td>
<td>1.09</td>
<td>.38</td>
</tr>
<tr>
<td>10000</td>
<td>390</td>
<td>.06</td>
<td>1.09</td>
<td>.38</td>
</tr>
<tr>
<td>10000</td>
<td>395</td>
<td>.06</td>
<td>1.09</td>
<td>.38</td>
</tr>
<tr>
<td>10000</td>
<td>400</td>
<td>.06</td>
<td>1.09</td>
<td>.38</td>
</tr>
<tr>
<td>10000</td>
<td>405</td>
<td>.06</td>
<td>1.09</td>
<td>.38</td>
</tr>
<tr>
<td>10000</td>
<td>410</td>
<td>.06</td>
<td>1.09</td>
<td>.38</td>
</tr>
<tr>
<td>10000</td>
<td>415</td>
<td>.06</td>
<td>1.09</td>
<td>.38</td>
</tr>
<tr>
<td>10000</td>
<td>420</td>
<td>.06</td>
<td>1.09</td>
<td>.38</td>
</tr>
<tr>
<td>10000</td>
<td>425</td>
<td>.06</td>
<td>1.09</td>
<td>.38</td>
</tr>
<tr>
<td>10000</td>
<td>430</td>
<td>.06</td>
<td>1.09</td>
<td>.38</td>
</tr>
<tr>
<td>10000</td>
<td>435</td>
<td>.06</td>
<td>1.09</td>
<td>.38</td>
</tr>
<tr>
<td>10000</td>
<td>440</td>
<td>.06</td>
<td>1.09</td>
<td>.38</td>
</tr>
<tr>
<td>10000</td>
<td>445</td>
<td>.06</td>
<td>1.09</td>
<td>.38</td>
</tr>
<tr>
<td>10000</td>
<td>450</td>
<td>.06</td>
<td>1.09</td>
<td>.38</td>
</tr>
<tr>
<td>10000</td>
<td>455</td>
<td>.06</td>
<td>1.09</td>
<td>.38</td>
</tr>
<tr>
<td>10000</td>
<td>460</td>
<td>.06</td>
<td>1.09</td>
<td>.38</td>
</tr>
<tr>
<td>10000</td>
<td>465</td>
<td>.06</td>
<td>1.09</td>
<td>.38</td>
</tr>
<tr>
<td>10000</td>
<td>470</td>
<td>.06</td>
<td>1.09</td>
<td>.38</td>
</tr>
<tr>
<td>10000</td>
<td>475</td>
<td>.06</td>
<td>1.09</td>
<td>.38</td>
</tr>
<tr>
<td>10000</td>
<td>480</td>
<td>.06</td>
<td>1.09</td>
<td>.38</td>
</tr>
<tr>
<td>10000</td>
<td>485</td>
<td>.06</td>
<td>1.09</td>
<td>.38</td>
</tr>
<tr>
<td>10000</td>
<td>490</td>
<td>.06</td>
<td>1.09</td>
<td>.38</td>
</tr>
<tr>
<td>10000</td>
<td>495</td>
<td>.06</td>
<td>1.09</td>
<td>.38</td>
</tr>
<tr>
<td>10000</td>
<td>500</td>
<td>.06</td>
<td>1.09</td>
<td>.38</td>
</tr>
</tbody>
</table>

Figure 9. Elapsed Time as a Function of F for Distribution 2 (Listing 13).
SUBROUTINE SORT(NUM,NO)
C IT IS MORE EFFICIENT TO PASS DATA THROUGH COMMON
  10 IF(I .LE. I) RETURN
      A = NO
      K = A * 381
      IF(MOD(I, 2) .EQ. 0) I = I + 1
  15 J = 0
      IF(J .GT. K) GO TO 10
      NUM2 = NUM(I + L)
      NUM1 = NUM(L)
      IF(NUM1 .LE. NUM2) GO TO 15
      NUM(I + L) = NUM1
      IF(L .LE. I) GO TO 20
      GO TO 25
      END
C LISTING 11

SUBROUTINE SORT(NUM,NO)
C IT IS MORE EFFICIENT TO PASS DATA THROUGH COMMON
  10 IF(I .LE. I) RETURN
      A = NO
      K = A * 381
      IF(MOD(I, 2) .EQ. 0) I = I + 1
  15 J = 0
      IF(J .GT. K) GO TO 10
      NUM2 = NUM(I + L)
      NUM1 = NUM(L)
      IF(NUM1 .LE. NUM2) GO TO 15
      NUM(I + L) = NUM1
      IF(L .LE. I) GO TO 20
      GO TO 25
      END
C LISTING 12
For each primary comparison, there is a chance of a primary exchange which will be followed generally by a secondary comparison, etc. Figure 6 indicates that secondary exchanges will be substantial for useful values of F; therefore, the expression for the comparisons of the DISTRIBUTION or PRIME sort will be that for PDQ sort, plus a series of terms involving probabilities which represent the various orders of exchanges (primary, secondary, etc.). Sort times for large NO may be expected to be proportional to Log NO for the PDQ sort, and to increase faster than a Log NO rate for PRIME sort.

PERFORMANCE EVALUATION

For a comparative evaluation of sort performance, some established sort algorithms published in the COMMUNICATIONS of the ACM4-7 were used. These algorithms were adapted for the sake of uniform notation and style. In addition, the subroutine SIFTUP of TREESORT3 was coded in-line to reduce the substantial overhead involved in the frequent calls to this procedure. The sort designated SINGLETON is one of the faster, more stable members of the QUICKSORT family.

An attempt to evaluate the efficiency of a procedure by frequency counts of critical parameters is not entirely satisfactory, even for differing versions of that procedure on the same machine. The particular form of a procedure is very important. Machine independent comparisons between differing procedures are even more difficult. Some of the results were omitted from the graphical sort performance data when interference between curves obscured the comparisons. Each curve has a label that refers to the listing of the code used in generating the data. (e.g., PDQ I L18 means PDQ INSERT #2, Listing 18.) The test results indicate that the PRIME and DISTRIBUTION sorts compare favorably overall to the QUICKSORT; and in the case of random distributions, the results increasingly favor the PDQ, PRIME, and DISTRIBUTION sorts as the array size decreases. The DISTRIBUTION sort is a compact and efficient sort suitable for in-line code applications because it is generally understandable; therefore, it may be modified easily for particular uses.
SUBROUTINE SORT(NUM, NO)
DIMENSION NUM(NO)
C
ACOPIATION OF SHOBSORT (ACM ALGORITHM 281 BY J. BORTHRODY)
COMMUNICATIONS OF THE ACM - VOLUME 17 / NUMBER 3 / MARCH, 1974
C
5 IF(I .LE. NO) GO TO 5
10 IF(I .LE. NO - 1) RETURN
K = NO - I
DO 20 J = 1, K
L = J - 1
IF(L .LE. 0) RETURN
H = NUM(J)
M = NUM(L)
MAX = NUM(M)
IF(NUM(L) .LE. NUM(M)) GO TO 20
MAX = NUM(M)
LISTING 14
20 CONTINUE
GO TO 5
C
SUBROUTINE SIFTUP(I)
DIMENSION NUM(NO)
C
ADAPTATION OF TREESORT3 (ACM ALGORITHM 245 BY R. W. FLOYD)
COMMUNICATIONS OF THE ACM - VOLUME 17 / NUMBER 3 / MARCH, 1974
DIMENSION NUM(NO)
NO = NO
I = NO/2 + 1
IF(L .LE. NO) GO TO 35
SUBROUTINE SIFTUP
C
20 IF(J .GT. NO) GO TO 30
IF(J .EQ. NO) GO TO 55
IF(NUM(J) .LE. NUM(J+1)) J = J + 1
GO TO 20
NUM(I) = NUM(J)
30 NUM(I) = MNNN
GO TO 10
C
45 IF(L .LE. NO) GO TO 60
SUBROUTINE SIFTUP
C
45 IF(J .GT. NO) GO TO 60
IF(J .EQ. NO) GO TO 85
IF(NUM(J) .LE. NUM(J+1)) J = J + 1
GO TO 45
NUM(I) = NUM(J)
60 NUM(I) = MNNN
MNNN = NUM(1)
NUM(1) = NUM(L)
NUM(L) = MNNN
GO TO 40
C
END
LISTING 15
SUBROUTINE SORT(NUM, NC)

DIMENSION IL(17), IU(17), NUM(NO)

IF (I .GE. J) GO TO 70
IF (NUM(I) .LE. NT) NUM(I) = NUM(IJ)
NT = NUM(IJ)
IF (NUM(I) .LE. NT) GO TO 40
NUM(I) = NUM(IJ)
NT = NUM(IJ)
GO TO 40
NUM(I) = NUM(IJ)
NT = NUM(IJ)
GO TO 40
NUM(I) = NUM(IJ)
NT = NUM(IJ)
GO TO 40
NUM(I) = NUM(IJ)
NT = NUM(IJ)
GO TO 40
NUM(L) = NUM(K)
NUM(K) = NT
IF (NUM(L) .LT. NT) GO TO 40
NUM(L) = NUM(K)
NT = NUM(K)
GO TO 40
L = L - 1
IF (L .LE. 1) GO TO 60
IL(M) = K
IU(M) = J
M = M + 1
GO TO 80
70 IF (M .EQ. 0) RETURN
80 IF (I .GE. 1) GO TO 10
90 IF (I .EQ. 1) GO TO 9
100 IF (I .EQ. J) GO TO 76
110 IF (NUM(I) .LE. NT) GO TO 90
120 IF (NUM(I) .LE. NT) GO TO 166
130 IF (NUM(K) .EQ. K) GO TO 96
GO TO 96
END

LISTING 16
SUBROUTINE SORT(NUM,NO)
C DIMENSION NUM(0)
C
10 A = NO
   IF (I .LE. 0) RETURN
   K = NO - I
   DO 15 J = 1,K
      NUM = NUM(I+J)
      IF (NUM .LT. NUM1+J) GO TO 15
      NUM1+J = NUM1
      NUM1 = NUM1+J
15 CONTINUE
C END

C
C SUBROUTINE SORT(NUM,NO)
C DIMENSION NUM(0)
C
10 A = 40
   IF (I .LE. 40) GO TO 10
   IF (I .GT. 9 AND MOD(I,2) .EQ. 0) I = I + 1
   K = 40 - 1
   DO 15 J = 1,K
      NUM = NUM(I+J)
      IF (NUM .LT. NUM1+J) GO TO 15
      NUM1+J = NUM1
      NUM1 = NUM1+J
15 CONTINUE
C END

C
C SUBROUTINE SORT(NUM,NO)
C DIMENSION NUM(0)
C
10 A = 40
   IF (I .LE. 40) GO TO 10
   IF (I .GT. 9 AND MOD(I,2) .EQ. 0) I = I + 1
   K = 40 - 1
   DO 15 J = 1,K
      NUM = NUM(I+J)
      IF (NUM .LT. NUM1+J) GO TO 15
      NUM1+J = NUM1
      NUM1 = NUM1+J
15 CONTINUE
C END

C
C ALTERNATE COMPARISONS
C IF THE ARRAY NUM IS IMMEDIATELY PRECEDED BY THE SMALLEST
C NUMBER PROCEED AS IN THE MAIN CODE; IF THE STATEMENT
C IF (L .GT. 0) GO TO 15 CAN BE REPLACED BY GO TO 35
C
C END
IT IS MORE EFFICIENT TO PASS DATA THROUGH COMMON

PRIME SORT

DATA INK /1,3,11,37,127,409,1373,4567,15241,50821,169331/

F = .30

DO 5 IN = 1,11
   IF (NO .LE. INK(IN))) GO TO 10
   CONTINUE
   10 IN = IN - 1
   IF (IN .LE. 0) RETURN
   K = NO - 1
   DO 20 J = 1,K
       NUM1 = NUM(I + L)
       NUM2 = NUM(I + L)
       IF (NUM1 .LE. NUM2) GO TO 20
       NUM1 = NUM2
       NUM(I + L) = NUM1
   20 CONTINUE
   GO TO 10

END

LISTING 19

SUBROUTINE SORT

COMMON TIER7(),NO,NUM(90W06)

DIMENSION INK(111)

PRIME SORT

DATA INK /1,3,11,37,127,409,1373,4567,15241,50821,169331/

F = .30

DO 5 IN = 1,11
   IF (NO .LE. INK(IN))) GO TO 10
   CONTINUE
   10 IN = IN - 1
   IF (IN .LE. 0) RETURN
   K = NO - 1
   DO 20 J = 1,K
       NUM1 = NUM(I + L)
       NUM2 = NUM(I + L)
       IF (NUM1 .LE. NUM2) GO TO 20
       NUM1 = NUM2
       NUM(I + L) = NUM1
   20 CONTINUE
   GO TO 10

END

LISTING 20
RANDOM ARRAYS

SECONDS

NO $\times 10^1$

28
RANDOM ARRAYS

<table>
<thead>
<tr>
<th>SECONDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
</tr>
<tr>
<td>0.25</td>
</tr>
<tr>
<td>0.50</td>
</tr>
<tr>
<td>0.75</td>
</tr>
<tr>
<td>1.00</td>
</tr>
<tr>
<td>1.25</td>
</tr>
<tr>
<td>1.75</td>
</tr>
<tr>
<td>2.00</td>
</tr>
</tbody>
</table>

NO \( \times 10^2 \)

- TREE 3 L15
- SINGLETON
- PRIME L19
- PDQ I L10
SORTED ARRAYS
SORTED BLOCKS
(Block Length = 64)
CONSTANT VALUE

- SINGELETON
- FOG 1 L10
- PRIME L19
- TREE3 L15

SECONDS

0.00 0.25 0.50 0.75 1.00 1.25 1.50 1.75 2.00

NO *10^2

0.00 20.00 40.00 60.00 80.00 100.00
CONSTANT VALUE

SECONDS
12.00
9.00
6.00
3.00
0.00

NO
0.00
15.00
30.00
45.00
60.00
75.00
90.00

PDS L17
PDS L16
SINGLETON
PDS IN L18
SHELL L14
PRIME L18
PRIMEC L20
TREES L18
DISTRIBUTION LIST

Aeronautical Systems Division (AFSC)
Wright-Patterson AFB, OH 45433
Attn: ASD/ENESH (P. T. Marth)
Attn: ASD/ENFTV (D. J. Wallick) (2 copies)
Attn: ASD/XROL (F. Campanile)
Attn: ASD/XROL (R. K. Frick)
Attn: ASD/XROT (G. B. Bennett)
Attn: ASD/XRE (S. E. Tate)
Attn: ASD/YPES (C. Gebhard)

Aerospace Medical Research Laboratories
AMRL/MEA
Area B, Bldg. 33
Wright-Patterson AFB, OH 45433
Attn: AMRL/MEA (CAPT G. J. Valentino)

Aerospace Rescue and Recovery Service (MAC)
Scott AFB, IL 62225
Attn: ARRS/DOQ (CAPT J. J. Draham, Jr.)

Air Force Acquisition Logistics Division
Wright-Patterson AFB, OH 45433
Attn: AFALD/PTEA (MAJ D. Waltman)

Air Force Avionics Laboratory
Wright-Patterson AFB, OH 45433
Attn: AFAL/WRA-1 (E. Leaphart)
Attn: AFAL/WRP (W. F. Bahret)

Air Force Flight Dynamics Laboratory
Wright-Patterson AFB, OH 45433
Attn: AFFDL/FES (CDIC) (2 copies)
Attn: AFFDL/FES (C. W. Harris)
Attn: AFFDL/FES (J. Hodges)
Attn: AFFDL/FES (R. W. Lauzze)
Attn: AFFDL/FES (D. W. Voyls)
Attn: AFFDL/FGL
Attn: AFFDL/TST (Library)

Air Force Logistic Command
Wright-Patterson AFB, OH 45433
Attn: AFLC/LOE (Commander)

Air Force Systems Command
Andrews AFB, DC 20334
Attn: AFSC/DLCAA (P. L. Sandler)
Air Force Test and Evaluation Center
Kirtland AFB, NM 87115
   Attn: AFTEC/OAR (K. Campbell)
   Attn: AFTEC/OAR (R. Blankert)

Air Force Weapons Laboratory
Kirtland AFB, NM 87117
   Attn: AFWL/PGV (CAPT J. K. Carson)
   Attn: AFTEC/OA (MAJ H. Rede)

Applied Technology Laboratory
Army Research & Technology Laboratory (AVRADCOM)
Ft. Eustis, VA 23604
   Attn: DAVDL-ATL-AL (Mr. Merritt)
   Attn: DAVDL-ATL-ASV (S. Pociluyko)
   Attn: DAVDL-ATL-ASV (H. W. Holland)
   Attn: DAVDL-ATL-ASV (J. T. Robinson)

Armament Development and Test Center
Eglin AFB, FL 32542
   Attn: ADTC/DLODL (Technical Library)
   Attn: ADTC/XR (C. T. Maney)
   Attn: ADTC/XRSP (E. Blair)

Armament Research and Development Command
Dover Base
Dover, NJ 07801
   Attn: DRDAR-LCS (S. K. Einbinder)

Army Aviation Research & Development Command
P.O. Box 209
St. Louis, MO 63166
   Attn: DRCPM-ASE-TM (MAJ Schwend) (2 copies)

Army Ballistic Research Laboratory
Aberdeen Proving Ground, MD 21005
   Attn: DRDAR-BLV (D. W. Mowrer)

Army Foreign Science and Technology Center
220 Seventh St., NE
Charlottesville, VA 22901
   Attn: DRXST-BA3 (E. R. McInturff)
   Attn: DRXST-CA2 (J. M Blake) (2 copies)

Army Materials and Mechanics Research Center
Watertown, MA 02172
   Attn: DRXMR-ER (F. C. Quigley)
   Attn: DRXMR-EM (C. F. Hickey, Jr.)
   Attn: DRXMR-PL (M. M. Murphy) (2 copies)
   Attn: DRXMR-XC (E. S. Wright)
Naval War College  
Newport, RI 02840  
Attn: Code E-111 (President)

Naval Weapons Center  
China Lake, CA 93555  
Attn: Code 317 (M. H. Keith)  
Attn: Code 3181 (C. Padgett) (2 copies)  
Attn: Code 3183 (C. Driussi)

Naval Weapons Engineering Support Activity  
Systems Analysis Dept.  
Bldg. 210-2 (ESA-19)  
Washington Navy Yard  
Washington, DC 20374  
Attn: Code ESA-1923 (C. W. Stokes III) (2 copies)  
Attn: Code 11621 (J. Stasko)

Naval Weapons Support Center  
Crane, IN 47522  
Attn: Code 502 (N. L. Papke)

Office of Naval Research  
800 N. Quincy Street  
Arlington, VA 22217  
Attn: ONR 474 (N. Perrone)

Pacific Missile Test Center  
Point Mugu, CA 93042  
Attn: Code 4253-3 (Naval Air Station, Technical Library) (2 copies)

Air Force Logistics Center  
Robins AFB, GA 31098  
Attn: WRALC/MMETE (LT W. Shelton)

Aerojet ElectroSystems Company  
A Div. of Aerojet-General Corp.  
1100 W. Hollyvale Street  
Azusa, CA 91702  
Attn: A. R. Moorman

AiResearch Manufacturing Co. of Arizona  
A Division of the Garrett Corp.  
P.O. Box 5217  
Phoenix, AZ 85010  
Attn: G. L. Merrill
Armament Systems, Inc.
712-F North Valley Street
Anaheim, CA 92801
Attn: J. Musch

A. T. Kearney and Company, Inc.
100 South Wacker Drive
Chicago, IL 60606
Attn: R. H. Rose

AVCO
Lycoming Division
550 So. Main St.
Stratford, CT 06497
Attn: R. Cuny

Battelle Memorial Institute
505 King Ave.
Columbus, OH 43201
Attn: J. H. Brown, Jr.

The BDM Corp.
2600 Yale Blvd SE.
Albuquerque, NM 87106
Attn: A. J. Holten

The BDM Corp.
1920 Aline Ave.
Vienna, VA 22180
Attn: J. W. Milenski

Beech Aircraft Corp.
9709 E. Central Ave.
Wichita, KS 67201
Attn: Engineering Library

Bell Helicopter Textron
Division of Textron Inc.
P.O. Box 482
Fort Worth, TX 76101
Attn: Security/Dept. 12, J. R. Johnson

The Boeing Aerospace Company
P.O. Box 3999
Seattle, WA 98124
Attn: J. G. Avery, M/S 41-37
Attn: R. J. Helzer, M/S 47-28
Falcon Research and Development Co.
2350 Alamo Ave., SE
Albuquerque, NM 87106
Attn: W. L. Baker

Falcon Research and Development Co.
696 Fairmount Ave.
Baltimore, MD 21204
Attn: J. A. Silva

Firestone Tire & Rubber Co.
Firestone Coated Fabric Co. Division
P.O. Box 869
Magnolia, AR 71753
Attn: D. L. Byerley

Ford Aerospace and Communications Corp.
Ford Road, P.O. Box A
Newport Beach, CA 92663
Attn: Library

General Dynamics Corp.
Convair Division
P.O. Box 80986
San Diego, CA 92138
Attn: Research Library, MZ 40-6540 (U. J. Sweeney)

General Dynamics Corp.
Fort Worth Division
Grants Lane, P.O. Box 748
Fort Worth, TX 76101
Attn: F. R. deTonnancour/G. W. Bowen

General Electric Co.
Aircraft Engine Group
1000 Western Ave.
West Lynn, MA 01910
Attn: J. M. Wannemacher

General Electric Co.
Aircraft Engine Business Group
Evendale Plant
Mail Drop H-9
Cincinnati, OH 45215
Attn: AEG Technical Information Center (J. J. Brady)
Attn: B. Alexander (2 copies)
General Research Corporation
P.O. Box 6770
5383 Hollister Ave.
Santa Barbara, CA 93111
   Attn: J. H. Cunningham
   Attn: R. Rodman

General Research Corporation
SWL Division, Suite 700, Park Place
7926 Jones Branch Dr.
McLean, VA 22101
   Attn: T. King

Goodyear Aerospace Corp.
Engineered Fabrics Division
1210 Massillon Rd.
Akron, OH 44315
   Attn: Library, D/152G (R. L. Vittitoe/J. R. Wolfersberger) (3 copies)

Grumman Aerospace Corp.
South Oyster Bay Rd.
Bethpage, NY 11714
   Attn: J. P. Archey Jr., Dept. 662, Mail C42-05
   Attn: R. W. Harvey, Mail C27-05
   Attn: H. L. Henze, B27-07
   Attn: J. Noack, M/S B10-25
   Attn: Technical Information Center, Plant 35 L01-35 (H. B. Smith)

Hughes Helicopters
A Division of Summa Corp.
Centinela Ave. & Teal St.
Culver City, CA 90230
   Attn: Library, 2/T2124 (D. K. Goss)

Institute for Defense Analyses
400 Army-Navy Drive
Arlington, VA 22202
   Attn: Technical Information Center, DIMO (P. Okamoto)

IIT Research Institute
10 West 35 Street
Chicago, IL 60616
   Attn: I. Pincus

The Johns Hopkins University
Applied Physics Laboratory
Johns Hopkins Road
Laurel, MD 20810
   Attn: C. F. Meyer
   Attn: V. W. Woodford
Kaman Aerospace Corporation
Old Windsor Rd.
Bloomfield, CT 06002
Attn: H.E. Showalter

Lockheed-California Co.
A Division of Lockheed Aircraft Corp.
2555 Hollywood Way
P.O. Box 551
Burbank, CA 91520
Attn: Technological Information Center, 84-40 Unit 35, Plant A-1
Attn: G. E. Raymer, D/75-84 Bldg. 63 A-1 (2 copies)
Attn: A. D. Jackmond, Dept. 75-60, Bldg. 170 B-1

Lockheed-Georgia Co.
A Division of Lockheed Aircraft Corp.
86 S. Cobb Drive
Marietta, GA 30063
Attn: W. T. Mikolowsky, 72-08 Zone 415
Attn: Sci-Tech Info Center, 72-34 Zone 26 (T. J. Kopkin)

Lockheed Missiles & Space Company, Inc.
A Subsidiary of Lockheed Aircraft Corp.
P.O. Box 504
Sunnyvale, CA 94088
Attn: G. R. Evans (5501-572-5)

Martin Marietta Corp.
Orlando Division
P.O. Box 5837
Orlando, FL 32855
Attn: Library (M. C. Griffith, MP-30)

McDonnell Douglas Corp.
Douglas Aircraft Company
3855 Lakewood Blvd.
Long Beach, CA 90846
Attn: Technical Library, Cl-250/36-84 AUTO 14-78 (3 copies)

McDonnell Douglas Corp.
P.O. Box 516
St. Louis, MO 63166
Attn: R. D. Detrich, Dept. 022

New Mexico Institute of Mining and Technology
Campus Station
Socorro, NM 87801
Attn: Tera
Northrop Corp.
Aircraft Division
3901 W. Broadway
Hawthorne, CA 90250
  Attn: J. H. Bach, 2130/83
  Attn: V. B. Bertagna, 3451/81
  Attn: H. W. Jones, 3360/82
  Attn: J. F. Paris, 3628/83

Northrop Corp.
Ventura Division
1515 Rancho Conejo Blvd.
P.O. Box 2500
Newbury Park, CA 91320
  Attn: M. Raine

The Rand Corp.
1700 Main St.
Santa Monica, CA 90406
  Attn: N. W. Crawford

R&D Associates
P.O. Box 9695
Marina Del Rey, CA 90291
  Attn: B. Jaeger

Rockwell International Corp.
Los Angeles Division
International Airport
5701 W. Imperial Hwy
Los Angeles, CA 90009
  Attn: R. Hurst, MB56
  Attn: W. L. Jackson
  Attn: S. C. Mellin, AD-25
  Attn: R. Moonan, AB78 (2 copies)
  Attn: T. C. Getten, MB56

Rockwell International Corp.
4300 E. Fifth Ave.
P.O. Box 1259
Columbus, OH 43216
  Attn: Technical Information Center (D. Z. Cox) (2 copies)

Science Applications, Inc.
200 Lomas Blvd., N. W., Suite 1020
Albuquerque, NM 87102
  Attn: Library
**ABSTRACT CARD**

Aeronautical Systems Division

PRIME and PDQ Sorts – Efficient Minimal Storage Sorting Algorithms, by R. R. Hilbrand, Wright-Patterson AFB, OH, ASD, for Joint Technical Coordinating Group/Aircraft Survivability, August 1979. 54 pp. (JTCG/AS-78-V-004, publication UNCLASSIFIED.)

One of the problems involved in computer programs for vulnerability assessment is that of rapidly sorting and arranging large sets of data. Two sorting algorithms, designated PRIME and PDQ, have been developed at ASD to more efficiently perform this function in vulnerability programs such

<table>
<thead>
<tr>
<th>(Over)</th>
<th>1 card, 8 copies</th>
</tr>
</thead>
</table>

Aeronautical Systems Division

PRIME and PDQ Sorts – Efficient Minimal Storage Sorting Algorithms, by R. R. Hilbrand, Wright-Patterson AFB, OH, ASD, for Joint Technical Coordinating Group/Aircraft Survivability, August 1979. 54 pp. (JTCG/AS-78-V-004, publication UNCLASSIFIED.)

One of the problems involved in computer programs for vulnerability assessment is that of rapidly sorting and arranging large sets of data. Two sorting algorithms, designated PRIME and PDQ, have been developed at ASD to more efficiently perform this function in vulnerability programs such

<table>
<thead>
<tr>
<th>(Over)</th>
<th>1 card, 8 copies</th>
</tr>
</thead>
</table>

Aeronautical Systems Division

PRIME and PDQ Sorts – Efficient Minimal Storage Sorting Algorithms, by R. R. Hilbrand, Wright-Patterson AFB, OH, ASD, for Joint Technical Coordinating Group/Aircraft Survivability, August 1979. 54 pp. (JTCG/AS-78-V-004, publication UNCLASSIFIED.)

One of the problems involved in computer programs for vulnerability assessment is that of rapidly sorting and arranging large sets of data. Two sorting algorithms, designated PRIME and PDQ, have been developed at ASD to more efficiently perform this function in vulnerability programs such

<table>
<thead>
<tr>
<th>(Over)</th>
<th>1 card, 8 copies</th>
</tr>
</thead>
</table>
as SESTEM and FASTGEN II. The results are compared to those obtained with three other algorithms, SHELLSORT, TREESORT3, and SINGLETON. The newly developed sorts are shown to be significantly faster on the ASD CDC 6600 computer than the existing sorts. When used in an ASD missile endgame model SESTEM, the average run time was reduced by 20 to 25%. Program listings, flow charts, and typical output data are presented.
One of the problems involved in computer programs for vulnerability assessment is that of rapidly sorting and arranging large sets of data. Two sorting algorithms, designated PRIME and PDQ, have been developed at ASD to more efficiently perform this function in vulnerability programs such...
as SESTEM and FASTGEN II. The results are compared to those obtained with three other algorithms, SHELLSORT, TREESORT3, and SINGLETON. The newly developed sorts are shown to be significantly faster on the ASD CDC 6600 computer than the existing sorts. When used in an ASD missile endgame model SESTEM, the average run time was reduced by 20 to 25%. Program listings, flow charts, and typical output data are presented.