

Algorithms



Neelam Akula

Spring 2021

Last Updated: March 23, 2021

Contents

1 Maximum Subarray Problem	3
1.1 Cubic Time	3
1.2 Quadratic Time	4
1.3 Linear Time	5
2 Bubble Sort	6
2.1 Pseudocode	6
2.2 Analysis of Comparisons	6
2.3 Analysis of Exchanges	6
3 Insertion Sort with Sentinel	8
3.1 Pseudocode	8
3.2 Analysis of Comparisons	8
3.3 Analysis of Exchanges	9
4 Insertion Sort without Sentinel	10
4.1 Pseudocode	10
4.2 Analysis of Comparisons	10
4.3 Analysis of Exchanges	11
5 Selection Sort	12
5.1 Pseudocode	12
5.2 Analysis of Comparisons	12
5.3 Analysis of Exchanges	12
6 Merge Sort	13
6.1 Pseudocode	13
6.2 Analysis of Merge	13
6.3 Analysis of Mergesort Comparisons	14
7 Heap Sort	16
7.1 Pseudocode	16
7.2 Create Heap	16
7.3 Finish Sort	18
7.4 Implementation	19
7.5 Optimization	19
8 Integer Arithmetic	20
8.1 Addition	20
8.2 Problem Size	20
8.3 Multiplication	20

9 Quicksort	23
9.1 Pseudocode	23
9.2 Analysis	23
9.3 Comments	27
A Comparison of Sorting Algorithms	30
A.1 Temporal and Spatial Locality	30
A.2 Comparison Table	30

1 Maximum Subarray Problem

Given an array of numbers (either positive or negative), we wish to calculate the subset of consecutive numbers whose sum is largest.

$$A : [3 \boxed{2} \boxed{-4} \boxed{-5} \boxed{6} \boxed{1} \boxed{-3} \boxed{7} \boxed{-8} \boxed{2}]$$

sums to 11

In this example, we see that the maximum sum is 11 with the indecies of the subarray being $A[5] : A[8]$.

1.1 Cubic Time

Pseudocode

Algorithm 1 Maximum Contiguous Sum (*Cubic*)

```

1:  $M \leftarrow 0$ 
2: for  $i = 1$  to  $n$  do
3:   for  $j = i$  to  $n$  do
4:      $S \leftarrow 0$ 
5:     for  $k = i$  to  $j$  do
6:        $S \leftarrow S + A[k]$ 
7:     end for
8:      $M \leftarrow \text{MAX}(M, S)$ 
9:   end for
10: end for
```

Analysis

From our algorithm above we can get the following analysis for it's complexity.

$$\begin{aligned}
 \sum_{i=1}^n \sum_{j=i}^n \sum_{k=i}^j 1 &= \sum_{i=1}^n \sum_{j=i}^n j - i + 1 \\
 &= \sum_{i=1}^n \sum_{j=1}^{n-i+1} j \\
 &= \sum_{i=1}^n \frac{(n-i+1)(n-i+2)}{2} \\
 &= \frac{1}{2} \sum_{i=1}^n (n-i+1)(n-i+2) \\
 &= \frac{1}{2} \sum_{n-i+1=1}^n i(i+1) \\
 &= \frac{1}{2} \frac{n(n+1)(n+2)}{3}
 \end{aligned}$$

From this, we can see this first version of our algorithm is $\Theta(n^3)$ or cubic time.

1.2 Quadratic Time

Pseudocode

Algorithm 1 Maximum Contiguous Sum (*Quadratic*)

```

1:  $M \leftarrow 0$ 
2: for  $i = 1$  to  $n$  do
3:    $S \leftarrow 0$ 
4:   for  $j = i$  to  $n$  do
5:      $S \leftarrow S + A[j]$ 
6:      $M \leftarrow \text{MAX}(M, S)$ 
7:   end for
8: end for

```

Analysis

We then use sums to analyze this algorithm.

$$\begin{aligned} \sum_{i=1}^n \sum_{j=i}^n 1 &= \sum_{i=1}^n n - i + 1 \\ &= \sum_{i=1}^n i \\ &= \frac{n(n+1)}{2} \end{aligned}$$

So this more efficient version is $\Theta(n^2)$ or quadratic time.

1.3 Linear Time

Using dynamic programming we can improve on this algorithm even further. Conceptually, the maximum sum is just the previous maximum sum plus the current value.

Pseudocode

Algorithm 1 Maximum Contiguous Sum (*Linear*)

```

1:  $M \leftarrow 0$ 
2:  $S \leftarrow 0$ 
3: for  $i = 1$  to  $n$  do
4:    $S \leftarrow \text{MAX}(S + A[i], 0)$ 
5:    $M \leftarrow \text{MAX}(M, S)$ 
6: end for

```

Analysis

It follows that this algorithm is $\Theta(n)$ or linear time. Correctness of this algorithm stems from proof by induction on $S \leftarrow \text{MAX}(S + A[i], 0)$, as this is the loop invariant.

2 Bubble Sort

2.1 Pseudocode

Algorithm 2 Bubble Sort

```

1: for  $i = n$  down to 2 do
2:   for  $j = 1$  to  $i - 1$  do
3:     if  $A[j] > A[j + 1]$  then
4:        $A[j] \leftrightarrow A[j + 1]$ 
5:     end if
6:   end for
7: end for

```

2.2 Analysis of Comparisons

Bubble Sort is designed in such a manner such that the state of the array to be listed does *not* change the number of comparisons it makes.

$$\begin{aligned}
\sum_{i=2}^n \sum_{j=1}^{i-1} 1 &= \sum_{i=2}^n i - 1 \\
&= \sum_{i=1}^{n-1} i \\
&= \frac{(n-1)n}{2} = \binom{n}{2}
\end{aligned}$$

2.3 Analysis of Exchanges

Worst Case

Worst case is when the list is reverse sorted, there will be the same number of exchanges as comparisons.

$$\frac{(n-1)n}{2}$$

Best Case

Best case is when the list is already sorted, in which there will be zero exchanges.

Average Case

To find the average case we must count the transpositions (two elements that are out of order related to one another). In best case there are no transpositions, and in worst case there are $\frac{(n-1)n}{2}$ transpositions. In a randomly permuted array each element is equally likely to be out of order so the total number of average case exchanges is half the comparisons.

$$\frac{1}{2} \cdot \frac{(n-1)n}{2} = \frac{(n-1)n}{4}$$

3 Insertion Sort with Sentinel

3.1 Pseudocode

Algorithm 3 Insertion Sort with Sentinel

```

1:  $A[0] \leftarrow -\infty$ 
2: for  $i = 2$  to  $n$  do
3:    $t \leftarrow A[i]$ 
4:    $j \leftarrow i - 1$ 
5:   while  $t < A[j]$  do
6:      $A[j + 1] \leftarrow A[j]$ 
7:      $j \leftarrow j - 1$ 
8:   end while
9:    $A[j + 1] \leftarrow t$ 
10: end for

```

3.2 Analysis of Comparisons

Worst Case

Worst case is when the array is reverse sorted, and every element must be moved. The while loop always decrements j to zero to compare against the sentinel value.

$$\sum_{i=2}^n i = \left(\sum_{i=1}^n i \right) - 1 = \frac{(n+1)n}{2} - 2 = \frac{(n+2)(n-1)}{2}$$

Best Case

Best case is when the array is already sorted, there is only one comparison for each iteration of the for loop.

$$\sum_{i=2}^n 1 = (n-2) + 1 = n - 1$$

Average Case

For average case we have to determine the probability that a given element will move. So we want the expected value of $\sum_{x \in X} P(x)V(x)$, where $P(x)$ is the prob-

ability that an element will end up a location and $V(x)$ is the number of moves.

$$\begin{aligned}
 \sum_{x \in X} P(x)V(x) &= \sum_{i=2}^n \sum_{j=1}^i \frac{1}{i} \cdot (i - j + 1) = \sum_{i=2}^n \frac{1}{i} \sum_{j=1}^i (i - j + 1) \\
 &= \sum_{i=2}^n \frac{1}{i} \sum_{j=1}^i j = \sum_{i=2}^n \frac{1}{i} \cdot \frac{(i+1)i}{2} \\
 &= \sum_{i=2}^n \frac{i+1}{2} = \frac{1}{2} \sum_{i=2}^n i + 1 \\
 &= \frac{1}{2} \sum_{i=1}^n i - 1 + (n-1) \\
 &= \frac{1}{2} \left(\frac{(n+1)n}{2} - 1 + \frac{(n-1)2}{2} \right) \\
 &= \frac{(n+4)(n-1)}{4}
 \end{aligned}$$

3.3 Analysis of Exchanges

Worst Case

Worst case is two moves for each iteration of outer loop plus worst case number of comparisons, minus the time when the comparison is false at the end. A shortcut of this is at each iteration we do one more move than comparison, so take the value we got above and add the number of loop iterations.

$$\frac{(n+2)(n-1)}{2} + n$$

Best Case

Best case is one initial move in the assignment on line 1 and then two moves during each iteration of the for loop. So we get,

$$1 + 2(n-1) = 2n - 1$$

Average Case

Average case is whenever there is a comparison there is a move except for the single instance in each iteration of when it evaluates to false. Thus the analysis method is the same as worst case.

$$\frac{(n+4)(n-1)}{4} + n$$

4 Insertion Sort without Sentinel

4.1 Pseudocode

Algorithm 4 Insertion Sort without Sentinel

```

1: for  $i = 2$  to  $n$  do
2:    $t \leftarrow A[i]$ 
3:    $j \leftarrow i - 1$ 
4:   while  $j > 0$  and  $A[j] > t$  do
5:      $A[j + 1] \leftarrow A[j]$ 
6:      $j \leftarrow j - 1$ 
7:   end while
8:    $A[j + 1] \leftarrow t$ 
9: end for
```

4.2 Analysis of Comparisons

Worst Case

Worst case is when the array is reverse sorted, and every i th iteration of the loop must compare against all previous $(i - 1)$ elements.

$$\begin{aligned}
 \sum_{i=2}^n \sum_{j=1}^{i-1} 1 &= \sum_{i=2}^n (i - 1) = \sum_{i=2}^n i - \sum_{i=2}^n 1 \\
 &= \frac{(n+1)n}{2} - (1 - (n-1)) \\
 &= \frac{(n-1)n}{2}
 \end{aligned}$$

Best Case

Best case is when the array is already sorted, so there will just be 1 comparison per iteration of the outermost loop. (Same as Insertion Sort with Sentinel.)

$$\sum_{i=2}^n 1 = (n - 2) + 1 = n - 1$$

Average Case

On average, the sentinel costs $\sum_{i=2}^n \frac{1}{i}$ comparisons. This is simply just the Harmonic Series. In other words, the sentinel costs $H_n - 1$ comparisons, so Insertion

Sort without Sentinel is

$$\frac{(n+4)(n-1)}{4} - (H_n - 1) \approx \frac{(n+4)(n-1)}{4} - \ln n$$

4.3 Analysis of Exchanges

Removing the sentinel adds no new exchanges so the best, worst, and average cases are all the same as Insertion Sort with Sentinel.

5 Selection Sort

5.1 Pseudocode

Algorithm 5 Selection Sort

```
1: for  $i = n$  down to 2 do
2:    $k \leftarrow 1$ 
3:   for  $j = 2$  to  $i$  do
4:     if  $A[j] > A[k]$  then
5:        $k \leftarrow j$ 
6:     end if
7:   end for
8:    $A[k] \leftrightarrow A[i]$ 
9: end for
```

5.2 Analysis of Comparisons

The number of comparisons is constant regardless of the state of the array.

$$\sum_{i=2}^n \sum_{j=2}^i 1 = \sum_{i=2}^n (i-1) = \frac{(n-1)}{n}$$

5.3 Analysis of Exchanges

Selection sort only performs one exchange per each iteration of the outermost loop, so there is a total of $n - 1$ exchanges.

6 Merge Sort

6.1 Pseudocode

Algorithm 6 Merge Sort

```

1: procedure MERGESORT( $A, p, r$ )
2:   if  $p < r$  then
3:      $q \leftarrow \lfloor (p+r)/2 \rfloor$ 
4:     MERGESORT( $A, p, q$ )
5:     MERGESORT( $A, q+1, r$ )
6:     MERGE( $A, (p, q), (q+1, r)$ )
7:   end if
8: end procedure

9: procedure MERGE( $A, (p, q), (q+1, r)$ )
10:  copy ( $A, p, r$ ) into ( $B, p, r$ )
11:   $i \leftarrow p$ 
12:   $j \leftarrow q+1$ 
13:   $k \leftarrow p$ 
14:  while  $i \leq q$  and  $j \leq r$  do
15:    if  $B[i] \leq B[j]$  then
16:       $A[k] \leftarrow B[i]$ 
17:       $i \leftarrow i + 1$ 
18:    else
19:       $A[k] \leftarrow B[j]$ 
20:       $j \leftarrow j + 1$ 
21:    end if
22:     $k \leftarrow k + 1$ 
23:  end while
24:  if  $i > q$  then
25:    copy ( $B, j, r$ ) into ( $A, k, r$ )
26:  else
27:    copy ( $B, i, q$ ) into ( $A, k, r$ )
28:  end if
29: end procedure

```

6.2 Analysis of Merge

Equal Size

Best case, we only do n comparisons (this is when $A_n < B_1$ or $B_n < A_1$). Worst case is $2n - 1$ comparisons (this is when A_n and B_n are the two largest elements).

The average case is $2n - 2 + \frac{2}{n+1}$.

Differing Sizes

Let subarray A be of size m and subarray B be of size n where $m \leq n$. Best case is m comparisons (this is when $A_m < B_1$). Worst case is $m + n - 1$ (this is when A_m and B_n are the two largest elements). When m is much smaller than n there are better algorithms we can use. For $m = 1$ binary search gives $\approx \lg n$.

Notes

Merging is *not* in place. It can be implemented in place but those algorithms are not practical.

6.3 Analysis of Mergesort Comparisons

We will analyze mergesort using the tree method, along with the assumption that n is a power of 2.

Problem Size	Tree	Comparisons
n	<img alt="A binary search tree diagram showing levels of nodes. The root node is at level 0. Level 1 has 2 nodes. Level 2 has 4 nodes. Level 3 has 8 nodes. Level 4 has 16 nodes. Level 5 has 32 nodes. Level 6 has 64 nodes. Level 7 has 128 nodes. Level 8 has 256 nodes. Level 9 has 512 nodes. Level 10 has 1024 nodes. Level 11 has 2048 nodes. Level 12 has 4096 nodes. Level 13 has 8192 nodes. Level 14 has 16384 nodes. Level 15 has 32768 nodes. Level 16 has 65536 nodes. Level 17 has 131072 nodes. Level 18 has 262144 nodes. Level 19 has 524288 nodes. Level 20 has 1048576 nodes. Level 21 has 2097152 nodes. Level 22 has 4194304 nodes. Level 23 has 8388608 nodes. Level 24 has 16777216 nodes. Level 25 has 33554432 nodes. Level 26 has 67108864 nodes. Level 27 has 134217728 nodes. Level 28 has 268435456 nodes. Level 29 has 536870912 nodes. Level 30 has 1073741824 nodes. Level 31 has 2147483648 nodes. Level 32 has 4294967296 nodes. Level 33 has 8589934592 nodes. Level 34 has 17179869184 nodes. Level 35 has 34359738368 nodes. Level 36 has 68719476736 nodes. Level 37 has 137438953472 nodes. Level 38 has 274877906944 nodes. Level 39 has 549755813888 nodes. Level 40 has 1099511627776 nodes. Level 41 has 2199023255552 nodes. Level 42 has 4398046511104 nodes. Level 43 has 8796093022208 nodes. Level 44 has 17592186044416 nodes. Level 45 has 35184372088832 nodes. Level 46 has 70368744177664 nodes. Level 47 has 140737488355328 nodes. Level 48 has 281474976710656 nodes. Level 49 has 562949953421312 nodes. Level 50 has 1125899906842624 nodes. Level 51 has 2251799813685248 nodes. Level 52 has 4503599627370496 nodes. Level 53 has 9007199254740992 nodes. Level 54 has 18014398509481984 nodes. Level 55 has 36028797018963968 nodes. Level 56 has 72057594037927936 nodes. Level 57 has 144115188075855872 nodes. Level 58 has 288230376151711744 nodes. Level 59 has 576460752303423488 nodes. Level 60 has 1152921504606846976 nodes. Level 61 has 2305843009213693952 nodes. Level 62 has 4611686018427387904 nodes. Level 63 has 9223372036854775808 nodes. Level 64 has 18446744073709551616 nodes. Level 65 has 36893488147419103232 nodes. Level 66 has 73786976294838206464 nodes. Level 67 has 147573952589676412928 nodes. Level 68 has 295147905179352825856 nodes. Level 69 has 590295810358705651712 nodes. Level 70 has 1180591620717411303424 nodes. Level 71 has 2361183241434822606848 nodes. Level 72 has 4722366482869645213696 nodes. Level 73 has 9444732965739290427392 nodes. Level 74 has 18889465931478580854784 nodes. Level 75 has 37778931862957161689568 nodes. Level 76 has 75557863725914323379136 nodes. Level 77 has 151115727458828646758272 nodes. Level 78 has 302231454917657293516544 nodes. Level 79 has 604462909835314587033088 nodes. Level 80 has 1208925819670629174066176 nodes. Level 81 has 2417851639341258348132352 nodes. Level 82 has 4835703278682516696264704 nodes. Level 83 has 9671406557365033392529408 nodes. Level 84 has 19342813114730066785058816 nodes. Level 85 has 38685626229460133570117632 nodes. Level 86 has 77371252458920267140235264 nodes. Level 87 has 154742504917840534280470528 nodes. Level 88 has 309485009835681068560941056 nodes. Level 89 has 618970019671362137121882112 nodes. Level 90 has 1237940039342724274243764224 nodes. Level 91 has 2475880078685448548487528448 nodes. Level 92 has 4951760157370897096975056896 nodes. Level 93 has 9903520314741794193950113792 nodes. Level 94 has 19807040629483588387900227584 nodes. Level 95 has 39614081258967176775800455168 nodes. Level 96 has 79228162517934353551600910336 nodes. Level 97 has 158456325035868707103201820672 nodes. Level 98 has 316912650071737414206403641344 nodes. Level 99 has 633825300143474828412807282688 nodes. Level 100 has 126765060028694965682561456576 nodes. Level 101 has 253530120057389931365122913152 nodes. Level 102 has 507060240114779862730245826304 nodes. Level 103 has 1014120480229559725460491652608 nodes. Level 104 has 2028240960459119450920983305216 nodes. Level 105 has 4056481920918238901841966610432 nodes. Level 106 has 8112963841836477803683933220864 nodes. Level 107 has 16225927683672955607367866441728 nodes. Level 108 has 32451855367345911214735732883456 nodes. Level 109 has 64903710734691822429471465766912 nodes. Level 110 has 129807421469383644858942931533824 nodes. Level 111 has 259614842938767289717885863067648 nodes. Level 112 has 519229685877534579435771726135296 nodes. Level 113 has 1038459371755069158871543452270592 nodes. Level 114 has 2076918743510138317743086904541184 nodes. Level 115 has 4153837487020276635486173809082368 nodes. Level 116 has 8307674974040553270972347618164736 nodes. Level 117 has 16615349948081106541944695236329472 nodes. Level 118 has 33230699896162213083889390472658944 nodes. Level 119 has 66461399792324426167778780945317888 nodes. Level 120 has 13292279958464885233555756189063576 nodes. Level 121 has 26584559916929770467111512378127152 nodes. Level 122 has 53169119833859540934223024756254304 nodes. Level 123 has 10633823966771908186844604951258808 nodes. Level 124 has 21267647933543816373689209802517616 nodes. Level 125 has 42535295867087632747378419605035232 nodes. Level 126 has 85070591734175265494756839210070464 nodes. Level 127 has 170141183468350530989513678420140928 nodes. Level 128 has 340282366936701061979027356840281856 nodes. Level 129 has 680564733873402123958054713680563712 nodes. Level 130 has 1361129467746804247916109427361127424 nodes. Level 131 has 2722258935493608495832218854722254848 nodes. Level 132 has 5444517870987216991664437709444509696 nodes. Level 133 has 1088903574197443398332875541888901932 nodes. Level 134 has 2177807148394886796665751083777803864 nodes. Level 135 has 4355614296789773593331502167555607728 nodes. Level 136 has 8711228593579547186663004335111215456 nodes. Level 137 has 17422457187159094373326008670222430912 nodes. Level 138 has 34844914374318188746652017340444861824 nodes. Level 139 has 69689828748636377493304034680889723648 nodes. Level 140 has 139379657497272754986608069361779457296 nodes. Level 141 has 278759314994545509973216138723558914592 nodes. Level 142 has 557518629989091019946432277447117829184 nodes. Level 143 has 1115037259978182039892864554894235658368 nodes. Level 144 has 2230074519956364079785729109788471316736 nodes. Level 145 has 4460149039912728159571458219576942633472 nodes. Level 146 has 8920298079825456319142916439153885266944 nodes. Level 147 has 17840596159650912638285832878307770533888 nodes. Level 148 has 35681192319301825276571665756615541067776 nodes. Level 149 has 71362384638603650553143331513231082135552 nodes. Level 150 has 142724769277207301106286663026462164271104 nodes. Level 151 has 285449538554414602212573326052924328542208 nodes. Level 152 has 570898577078829204425146652105848657084416 nodes. Level 153 has 1141797154157658408850293304211693114168832 nodes. Level 154 has 2283594308315316817700586608423386228337664 nodes. Level 155 has 4567188616630633635401173216846772456675328 nodes. Level 156 has 9134377233261267270802346433693544913350656 nodes. Level 157 has 18268754466522534541604692867387089826701312 nodes. Level 158 has 36537508933045069083209385734774179653402624 nodes. Level 159 has 73075017866090138166418771469548359306805248 nodes. Level 160 has 146150035732180276332835542939096718613610496 nodes. Level 161 has 292300071464360552665671085878193437227220992 nodes. Level 162 has 584600142928721005331342171756386874454441984 nodes. Level 163 has 116920028585744201066268434351273774890883968 nodes. Level 164 has 233840057171488402132536868702547549781767936 nodes. Level 165 has 467680114342976804265073737405095099563535872 nodes. Level 166 has 935360228685953608530147474810190199127071744 nodes. Level 167 has 1870720457371907217060294949620380398254143488 nodes. Level 168 has 3741440914743814434120589899240760796508286976 nodes. Level 169 has 7482881829487628868241179798481521593016573952 nodes. Level 170 has 14965763658975257736482359596963043186033147904 nodes. Level 171 has 29931527317950515472964719193926086372066295808 nodes. Level 172 has 59863054635901030945929438387852172744132591616 nodes. Level 173 has 119726109271802061891858876775704345488265833232 nodes. Level 174 has 239452218543604123783717753551408690976531666464 nodes. Level 175 has 478904437087208247567435507102817381953063332928 nodes. Level 176 has 957808874174416495134871014205634763906126665856 nodes. Level 177 has 1915617748348832990269542028411269527812253331712 nodes. Level 178 has 3831235496697665980539084056822539055624506663424 nodes. Level 179 has 7662470993395331961078168113645078111249013326848 nodes. Level 180 has 15324941986786663922156336227290156222498026653696 nodes. Level 181 has 30649883973573327844312672454580312444996053307392 nodes. Level 182 has 61299767947146655688625344909160624889992106614784 nodes. Level 183 has 122599535894293311377250689818321249779984213229568 nodes. Level 184 has 245199071788586622754501379636642499559968426459136 nodes. Level 185 has 490398143577173245509002759273284999119936852918272 nodes. Level 186 has 980796287154346491018005518546569998239873705836544 nodes. Level 187 has 1961592574308692982036011037093139996479747411672988 nodes. Level 188 has 3923185148617385964072022074186279992959494823345976 nodes. Level 189 has 7846370297234771928144044148372559985918989646691952 nodes. Level 190 has 1569274059446954385628808829674511997183797929338384 nodes. Level 191 has 3138548118893908771257617659349023994367595858676768 nodes. Level 192 has 6277096237787817542515235318698047988735191717353536 nodes. Level 193 has 12554192475575635085030470637396095977470383434707072 nodes. Level 194 has 25108384951151270170060941274792191954940766869414144 nodes. Level 195 has 50216769902302540340121882549584383909881533738828288 nodes. Level 196 has 100433539804605080680243765098168767819763067477656576 nodes. Level 197 has 200867079609210161360487530196337535639526134955313152 nodes. Level 198 has 401734159218420322720975060392675071278552269850626304 nodes. Level 199 has 803468318436840645441950120785350142557104539701252608 nodes. Level 200 has 1606936636873681290883900241570700285114209079402505216 nodes. Level 201 has 3213873273747362581767800483141400570228418158805010432 nodes. Level 202 has 6427746547494725163535600966282801140456836317610020864 nodes. Level 203 has 12855493094989450327071201932565602280913672635220041728 nodes. Level 204 has 25710986189978900654142403865131204561827345270440083456 nodes. Level 205 has 5142197237995780130828480773026240912365469054088016691 nodes. Level 206 has 10284394475991560261656961546052481825930938108176033382 nodes. Level 207 has 20568788951983120523313923092104963651861876216352066764 nodes. Level 208 has 41137577903966241046627846184209927303723752432704133528 nodes. Level 209 has 82275155807932482093255692368419854607447504865408267056 nodes. Level 210 has 164550311615864964186511384736839709214895009728016534112 nodes. Level 211 has 329100623231729928373022769473679418429790019456033068224 nodes. Level 212 has 658201246463459856746045538947358836859580038912066136448 nodes. Level 213 has 131640249292691971349209107789471767371916007782413263296 nodes. Level 214 has 263280498585383942698418215578943534743832015564826526592 nodes. Level 215 has 526560997170767885396836431157887069487664031129653053184 nodes. Level 216 has 1053121994341535770793672862315774138975328062259306056368 nodes. Level 217 has 2106243988683071541587345724631548277950656124518612112736 nodes. Level 218 has 4212487977366143083174691449263096555901312249037224225472 nodes. Level 219 has 8424975954732286166349382898526193111802624488074448450944 nodes. Level 220 has 16849951909464572332698765797052386223605248976148896901888 nodes. Level 221 has 33699903818929144665397531594104772447210497952297793803776 nodes. Level 222 has 67399807637858289330795063188209544884420995904595587607552 nodes. Level 223 has 13479961527571657866159012637641908976884199180918117521504 nodes. Level 224 has 26959923055143315732318025275283817953768398361836235043008 nodes. Level 225 has 53919846110286631464636050550567635907536796723672470086016 nodes. Level 226 has 107839692220573262929272101101135271815073593447344940172032 nodes. Level 227 has 215679384441146525858544202202270543630147186894689880344064 nodes. Level 228 has 431358768882293051717088404404541087260294373789379760688128 nodes. Level 229 has 862717537764586103434176808809082174520588747578759521376256 nodes. Level 230 has 1725435075529172206868353617618164349041177475157519042752512 nodes. Level 231 has 3450870151058344413736707235236328698082354950315038085505024 nodes. Level 232 has 6901740302116688827473414470472657396164709850630076161010048 nodes. Level 233 has 13803480604233377654946828940945314792329419701260152322020096 nodes. Level 234 has 27606961208466755309893657881890629484658839402520304644040192 nodes. Level 235 has 55213922416933510619787315763781258969317678805040609288080384 nodes. Level 236 has 110427844833867021239574631527562578938635357610081218576160768 nodes. Level 237 has 220855689667734042479149263055125157877270715220162437152321536 nodes. Level 238 has 441711379335468084958298526110250315754541430440324874304643072 nodes. Level 239 has 883422758670936169916597052220500631509082860880649488609286144 nodes. Level 240 has 176684551734187233983319410444100126301816572176129897721857288 nodes. Level 241 has 353369103468374467966638820888200252603633144352259795443714576 nodes. Level 242 has 706738206936748935933277641776400505207266288704519590887429152 nodes. Level 243 has 141347641387349787186655328355280101041453257740903918177855832 nodes. Level 244 has 282695282774699574373310656710560202082906515481807836355711664 nodes. Level 245 has 565390565549399148746621313421120404165813030	

number of comparisons is

$$\begin{aligned}\sum_{i=0}^{\lg n-1} 2^i \left(\frac{n}{2^i} - 1 \right) &= \sum_{i=0}^{\lg n-1} (n - 2^i) \\&= \sum_{i=0}^{\lg n-1} n - \sum_{i=0}^{\lg n-1} 2^i \\&= (n \lg n) - (2^{\lg n-1+1} - 1) \\&= (n \lg n) - (n - 1) \\&= n \lg n - n + 1\end{aligned}$$

As a recurrence,

$$\begin{aligned}T(n) &= T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + (n - 1) \\&= 2T\left(\frac{n}{2}\right) + n - 1, \quad T(0) = T(1) = 0\end{aligned}$$

7 Heap Sort

7.1 Pseudocode

Algorithm 7 Heap Sort

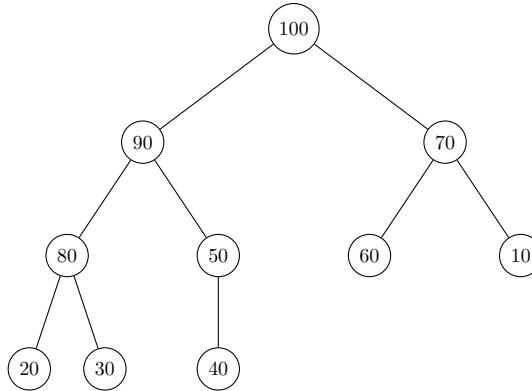
```

1: procedure HEAPSORT( $A, n$ )                                 $\triangleright A$  is a list and  $n$  is  $A$ 's size
2:   for  $r = \lfloor n/2 \rfloor$  down to 1 do                       $\triangleright$  Create Heap
3:     SIFT( $r, n$ )
4:   end for
5:
6:   for  $m = n$  down to 2 do                                 $\triangleright$  Finish Sort
7:      $A[1] \leftrightarrow A[m]$ 
8:     SIFT(1,  $m - 1$ )
9:   end for
10:  end procedure

11: procedure SIFT( $p, m$ )                                 $\triangleright p$  is the root and  $m$  is the size of the list
12:    $c \leftarrow 2p$ 
13:   while  $c \leq m$  do
14:     if  $c < m$  then
15:       if  $A[c + 1] > A[c]$  then
16:          $c \leftarrow c + 1$ 
17:       end if
18:     end if
19:
20:     if  $A[c] > A[p]$  then
21:        $A[p] \leftrightarrow A[c]$ 
22:        $p \leftarrow c$ 
23:        $c \leftarrow 2p$ 
24:     else
25:       exit while loop
26:     end if
27:   end while
28: end procedure
  
```

7.2 Create Heap

A Heap is a binary tree where every value is larger than its children. Equivalently its descendants. For the purposes of this class will we require that all binary trees are full binary trees.



The traditional way of creating a heap is to insert at the end of the array and sift up. Robert Floyd created a better algorithm for creating the heap. Treat the tree as a recursive heap: each parent is the parent of 2 heaps, and sift from the bottom up. Create heap on left, create heap on right, then sift root down, and move up a level.

Heap Creation Analysis

In a binary tree, most nodes are near the bottom, so when doing the bottom up technique, most of the work is done at the bottom. The number of comparisons can be calculated like so,

$$\begin{aligned} \frac{n}{2} \cdot 0 + \frac{n}{4} \cdot 2 + \frac{n}{8} \cdot 4 + \dots &= \\ = n \left[\frac{1}{2} + \frac{2}{4} + \frac{3}{8} + \dots \right] \end{aligned}$$

Note that $\left[\frac{1}{2} + \frac{2}{4} + \frac{3}{8} + \dots \right]$ can be written as so,

$$\begin{aligned} \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \dots &= 1 \\ \frac{1}{4} + \frac{1}{8} + \dots &= \frac{1}{2} \\ \frac{1}{8} + \dots &= \frac{1}{4} \end{aligned}$$

Which all together becomes

$$1 + \frac{1}{2} + \frac{1}{4} + \dots = 2$$

So Heap creation does $2n$ comparisons.

7.3 Finish Sort

After heap creation we then sort it,

- Put root node at the bottom of the array (it must be the largest element) so it goes to end of the sorted array.
- Then take the bottom right hand leaf and move to a temporary space.
- Then sift, reordering the tree and put the temporary leaf in its proper spot.
- Repeat until all elements are sorted.

Heap Sort Analysis

Each level has two comparisons, child and temporary. There are $\approx \lg n$ levels, so the total comparisons for a sift is $\approx 2 \lg n$. This is done for each element in the tree so worst case we get $\approx 2n \lg n$. But heap shrinks upon each iteration (it removes an element) so we take the sum from 0 to $n - 1$.

$$\begin{aligned}
\sum_{i=0}^{n-1} 2 \lg(i+1) &\approx 2 \sum_{i=1}^n \lg i \\
&= 2[\lg 1 + \lg 2 + \lg 3 + \cdots + \lg n] \\
&= 2 \lg(1 \times 2 \times 3 \times \cdots \times n) \\
&= 2 \lg(n!) \\
&= 2 \lg \left[\left(\frac{n}{e} \right)^n \cdot \sqrt{2\pi n} \right] \\
&\approx 2 \left[n \lg \left(\frac{n}{e} \right) + \frac{\lg(2\pi n)}{2} \right] \\
&= 2n \lg n - 2n \lg e + \lg n + \lg(2\pi) \\
&= 2n \lg n + O(n)
\end{aligned}$$

From this we see that even while the tree shrinks, it does not shrink fast enough to make some notable difference. We are still doing $2n \lg n$ comparisons.

7.4 Implementation

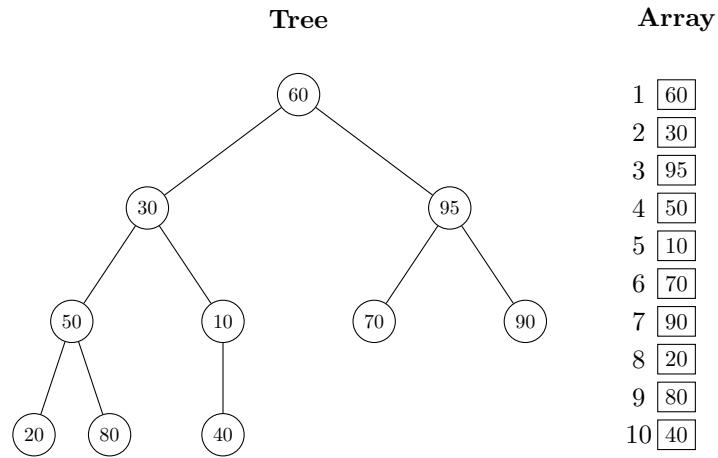


Figure 1: Use an array to implement a tree.
 Node has index i , Left child is $2i$, Right child is $2i + 1$, and Parent is $\lfloor \frac{i}{2} \rfloor$.

Create

The first parent is at index $\lfloor \frac{n}{2} \rfloor$. Start there and sift down during heap creation. Siblings can be reached by adding or subtracting 1. The result is a created heap.

Finish

To finish the sort we push bottom into tmp first, then move heap root into the bottom most spot.

7.5 Optimization

As Heapsort stands, the result is worse than merge sort. $\Theta(2n \lg n)$ vs $\Theta(n \lg n)$ shows us that much. We compare tmp against both children and this doubles our total number of comparisons. Instead we can sift the hole left by the root down to the bottom in $\lg n$ comparisons. Then put tmp in the hole and sift it back into position. It follows then that we give up $2n$ comparisons on average. Further optimization can be achieved by binary searching up. This gives Heapsort $n \lg n + n \lg(\lg n) = \Theta(n \lg n)$ performance.

8 Integer Arithmetic

8.1 Addition

As you would expect, the time to add two n -digit numbers is proportional to the number of digits, $\Theta(n)$. Since each digit must be used to compute the sum.

$$\begin{array}{r} & \overset{1}{\cancel{1}} \\ & 2 \\ & 3 \\ + & 4 \\ \hline & 8 \\ & 7 \\ \hline & 2 \\ & 1 \\ \hline 6 & 9 & 1 & 2 \end{array}$$

We assume that each digital addition takes constant time (with carry also) and label this constant α . The time to add two n -digit numbers is then αn .

8.2 Problem Size

If we were attempting to determine if a number was prime we would divide it by consecutive primes up to the square root of that number. In Computer Science we want to approach all problems in terms of the problem size. So we should look at both in terms of the *number of digits* and not the size of the number. So while the prime identification problem is $\Theta(\sqrt{m})$ where m is the number, expressed in binary as 2^n , n being the number of binary digits results in $\Theta(2^{n/2})$. Giving an exponential time algorithm.

8.3 Multiplication

The elementary approach to multiplication runs in $\Theta(n^2)$ time because every digit on top is multiplied by every digit on the bottom. There are $2n(n - 1)$ atomic additions in this elementary approach.

$$\begin{array}{r} \times \quad \overset{1}{\cancel{1}} \quad \overset{2}{\cancel{2}} \quad \overset{3}{\cancel{3}} \quad \overset{4}{\cancel{4}} \\ \quad \quad \quad \overset{5}{\cancel{5}} \quad \overset{6}{\cancel{6}} \quad \overset{7}{\cancel{7}} \quad \overset{8}{\cancel{8}} \\ \hline \quad \quad \quad 2 \\ \quad \quad \quad 4 \\ \quad \quad \quad 6 \\ \quad \quad \quad 8 \\ \quad \quad \quad 0 \\ \quad \quad \quad 4 \\ \quad \quad \quad 0 \\ \quad \quad \quad 0 \\ \hline 2 \\ 4 \\ 6 \\ 8 \\ 0 \\ 0 \\ 6 \\ 6 \\ 5 \\ 2 \end{array}$$

Recursive Multiplication

By memorizing numbers in large bases, say base 100, 2-digit decimal numbers can be treated as 1-digit numbers in base 100. That makes the multiplication easier, because there will be fewer steps. To generalize this for n -digit numbers, treat them as base $10^{n/2}$ and cut each in half. Each has $n/2$ digits, and we know the base.

Then, recurse through this method until we reach a base we have memorized. Then multiply them as two 2-digit numbers, and pull the answer.

Say we want to multiply two 2-digit numbers, ab and cd , we can then do the following.

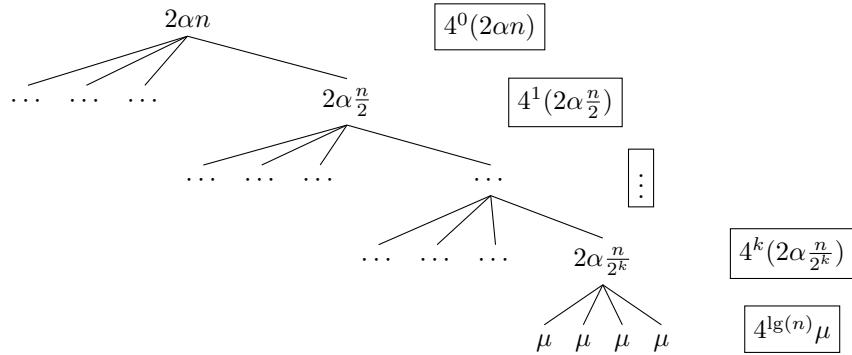
$$ad \times bc = ac + (ad + bc) + bd$$

In the example above ac is an n -digit number as are ad , bc , and bd . When adding, bc is not shifted, bc and ad are shifted by $n/2$ digits, and ac is shifted by n digits. Since ac and bd have no overlap you can add them through concatenation. Then, $ad + bc$ takes αn time and $ac + bd$ is also αn time. This results in a total add time of $2\alpha n$.

So without writing out a formal algorithm,

$$M(n) = 4M\left(\frac{n}{2}\right) + 2\alpha n$$

is the recursion we wish to work on. Now, to solve we assume that the base case time to multiply is μ . The base case is when both of our numbers are 1 digit long, i.e. $M(1) = \mu$.



Now to calculate the total number of atomic actions,

$$\begin{aligned}
 &= \sum_{i=0}^{\lg(n)-1} (4^i(2\alpha \frac{n}{2^i})) + 4^{\lg(n)}\mu \\
 &= 2\alpha n \sum_{i=0}^{\lg n - 1} \frac{4^i}{2^i} \dots \\
 &= 2\alpha n \sum_{i=0}^{\lg n - 1} 2^i \dots \\
 &= 2\alpha n (2^{\lg n - 1 + 1} - 1) \\
 &= 2\alpha n (n - 1) + 4^{\lg(n)}\mu \\
 &= 2\alpha n (n - 1) + n^{\lg(4)}\mu \\
 &= 2\alpha n (n - 1) + n^2\mu \\
 &= 2\alpha n (n - 1) + n^2\mu
 \end{aligned}$$

So we see that with this method we are doing n^2 multiplications and n^2 additions. In other words, this recursive method has the exact same running time as the elementary method.

Faster Multiplication

9 Quicksort

9.1 Pseudocode

Algorithm 8 Quicksort

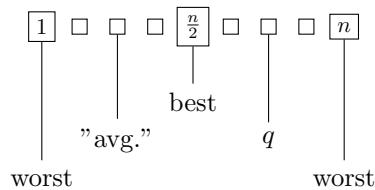
```

1: procedure QUICKSORT( $A, p, r$ )
2:   if  $p < r$  then
3:      $q \leftarrow \text{PARTITION}(A, p, r)$ 
4:     QUICKSORT( $A, p, q - 1$ )
5:     QUICKSORT( $A, q + 1, r$ )
6:   end if
7: end procedure

8: function PARTITION( $A, p, r$ )
9:    $X \leftarrow A[r]$ 
10:   $i \leftarrow p - 1$ 
11:  for  $j = p$  to  $r - 1$  do
12:    if  $A[j] \leq X$  then
13:       $i \leftarrow i + 1$ 
14:       $A[i] \leftrightarrow A[j]$ 
15:    end if
16:  end for
17:   $A[i + 1] \leftrightarrow A[r]$ 
18:  return ( $i + 1$ )
19: end function
  
```

9.2 Analysis

The worst case is when the pivot is either in the *first* or *last* index, best case is when the pivot is the *median* index, and the "average" case is when the pivot is between the first/last and median index (here q will represent the *true* average).



Worst Case

As a recurrence,

$$\begin{aligned} T(n) &= T(n-1) + n - 1, \quad T(0) = T(1) = 0 \\ &= \sum_{i=1}^{n-1} i = \frac{(n-1)n}{2} \end{aligned}$$

Best Case

As a recurrence,

$$\begin{aligned} T(n) &= 2T\left(\frac{n}{2}\right) + n - 1, \quad T(0) = T(1) = 0 \\ &= n \lg n - n + 1 \end{aligned}$$

Note: This is the same recurrence as Mergesort!

Average Case

As a recurrence,

$$T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{3n}{4}\right) + n - 1, \quad T(0) = T(1) = 0$$

Solve with Strong Constructive Induction.

Guess: $T(n) \leq an \lg n$ for $n \geq 1$ and some constant a .

Base case $n = 1$: $an \lg n = a1 \lg 1 = a \cdot 1 \cdot 0 = 0$, $T(1) = 0$ and $0 \leq 0$.

Inductive Hypothesis: Assume true for $< n$, $T(k) \leq ak \lg k$ for $1 \leq k \leq n$.

Inductive Step:

$$\begin{aligned} T(n) &= T\left(\frac{n}{4}\right) + T\left(\frac{3n}{4}\right) + n - 1 \\ &\leq a\frac{n}{4} \lg\left(\frac{n}{4}\right) + a\frac{3n}{4} \lg\left(\frac{3n}{4}\right) + n - 1 \quad \text{by IH} \\ &\leq a\frac{n}{4} (\lg n - \lg 4) + a\frac{3n}{4} (\lg 3 + \lg n - \lg 4) + n - 1 \\ &\leq \frac{an}{4} (\lg n - 2) + \frac{3an}{4} (\lg 3 + \lg n - 2) + n - 1 \\ &\leq \frac{an \lg n}{4} - \frac{2an}{4} + \frac{3an \lg 3}{4} + \frac{3an \lg n}{4} - \frac{2 \cdot 3an}{4} + n - 1 \\ &\leq an \lg n + \left(-\frac{a}{2} + \frac{3a \lg 3}{4} - \frac{3}{2}a + 1\right) n - 1 \\ &\leq an \lg n + \left(\left(\frac{3 \lg 3}{4} - 2\right)a + 1\right) n - 1 \end{aligned}$$

It follows that we need

$$\left(\frac{3\lg 3}{4} - 2\right)a + 1 \leq 0 \implies a \geq \frac{1}{2 - \frac{3\lg 3}{4}}$$

$$a \gtrsim 1.23$$

Thus,

$$T(n) \lesssim 1.23n \lg n$$

Exact Average Case

As a recurrence,

$$\begin{aligned} T(n) &= \left[\sum_{q=1}^n \frac{1}{n} (T(q-1) + T(n-q)) \right] + n - 1, \quad T(0) = T(1) = 0 \\ &= \frac{1}{n} \sum_{q=1}^n (T(q-1) + T(n-q)) + n - 1 \\ &= \frac{1}{n} \sum_{q=1}^n T(q-1) + \frac{1}{n} \sum_{q=1}^n T(n-q) + n - 1 \\ &= \frac{1}{n} \sum_{q=0}^{n-1} T(q) + \frac{1}{n} \sum_{q=0}^{n-1} T(n-q) + n - 1^* \\ &= \frac{2}{n} \sum_{q=0}^{n-1} T(q) + n - 1 \end{aligned}$$

*Note that:

$$\begin{aligned} \sum_{q=1}^n T(q-1) &= T(0) + T(1) + T(2) + \cdots + T(n-1) \\ \sum_{q=1}^n T(n-q) &= T(n-1) + T(n-2) + T(n-3) + \cdots + T(0) \end{aligned}$$

Solve with Strong Constructive Induction.

Guess: $T(n) \leq an \lg n$ for $n \geq 1$ and some constant a .

Base case $n = 1$: $an \lg n = a1 \lg 1 = a \cdot 1 \cdot 0 = 0$, $T(1) = 0$ and $0 \leq 0$.

Inductive Hypothesis: Assume true for n , $T(k) \leq ak \lg k$ for $1 \leq k \leq n$.

Inductive Step:

$$\begin{aligned}
 T(n) &= n - 1 + \frac{2}{n} \sum_{q=0}^{n-1} T(q) \\
 &= n - 1 + \frac{2}{n} \sum_{q=1}^{n-1} T(q) \\
 &\leq n - 1 + \frac{2}{n} \sum_{q=1}^{n-1} aq \lg q \quad \text{by IH} \\
 &\leq n - 1 + \frac{2a}{n} \int_1^n x \lg x dx \quad \text{by integral bound} \\
 &= n - 1 + \frac{2a}{n} \left[\frac{x^2 \lg x}{2} - \frac{x^2 \lg e}{4} \right] \Big|_1^n \\
 &= n - 1 + an \lg n - \frac{an \lg e}{2} + \frac{a \lg e}{2n} \\
 &= an \lg n + \left[1 - \frac{a \lg e}{2} \right] n - 1 + \frac{a \lg e}{2n}
 \end{aligned}$$

It follows that we need

$$1 - \frac{a \lg e}{2} \leq 0 \implies a \geq \frac{2}{\lg e}$$

So set $a = \frac{2}{\lg e} \approx 1.39$, we then need

$$\frac{a \lg e}{2n} - 1 \leq 0 \iff \frac{2 \lg e}{(\lg e) 2n} - 1 \leq 0 \iff \frac{1}{n} - 1 \leq 0$$

Thus,

$$T(n) \lesssim 1.39n \lg n$$

We can realize a more natural formula,

$$T(n) \leq an \lg n = \frac{2n \lg n}{\lg e} = 2n \ln n$$

Theorem

The expected number of comparisons for Quicksort is $\approx 2n \ln n$.

Blum's Exact Average Case

Let $P(i, j)$ be the probability that the i th smallest and j th smallest elements are compared.

Theorem

The average number of comparisons for quicksort is

$$\sum_{1 \leq i < j \leq n} P(i, j)$$

Only matters the first time an element from $A[i], \dots, A[j]$ is picked as pivot. The probability that the i th and j th smallest elements are compared during the execution of quicksort is:

$$\frac{2}{j - i + 1}$$

Then,

$$\begin{aligned} \sum_{1 \leq i < j \leq n} P(i, j) &= \sum_{i=1}^n \sum_{j=i+1}^n P(i, j) = \sum_{i=1}^n \sum_{j=i+1}^n \frac{2}{j - i + 1} \\ &= 2 \sum_{i=1}^n \sum_{j=i+1}^n \frac{1}{j - i + 1} \\ &= 2 \sum_{i=1}^n \left(\frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n-i+1} \right) \\ &= 2 \sum_{i=1}^n (H_{n-i+1} - 1) = 2 \sum_{i=1}^n H_{n-i+1} - 2 \sum_{i=1}^n 1 \\ &= 2(H_n + H_{n-1} + \dots + H_1) - 2 \sum_{i=1}^n 1 \\ &= 2 \sum_{i=1}^n H_i - 2n \\ &= 2((n+1)H_n - n) - 2n \\ &= 2(n+1)H_n - 4n \approx 2n \ln n \end{aligned}$$

9.3 Comments

Quicksort Code

Recall the Quicksort procedure,

```
procedure QUICKSORT( $A, p, r$ )
  if  $p < r$  then
     $q \leftarrow \text{PARTITION}(A, p, r)$ 
    QUICKSORT( $A, p, q - 1$ )
    QUICKSORT( $A, q + 1, r$ )
  end if
end procedure
```

When executing $\text{quicksort}(A, p, q - 1)$ we must stack up $\text{quicksort}(A, q + 1, r)$ to be executed later. We need to store the pair of index values $(q + 1, r)$.

Stack in Action

The left table represents the pivot being the largest element, and the right table represents the pivot being the smallest element.

Pivot	Stack	Pivot	Stack
$A[n]$	$(n+1, n)$	$A[1]$	$(2, n)$
$A[n-1]$	$(n, n-1)$	$A[2]$	$(3, n)$
$A[n-2]$	$(n-1, n-2)$	$A[3]$	$(4, n)$
\vdots	\vdots	\vdots	\vdots
$A[3]$	$(4, 3)$	$A[n-2]$	$(n-1, n)$
$A[2]$	$(3, 2)$	$A[n-1]$	(n, n)
$A[1]$		$A[n]$	

When the pivot is the smallest element the stack remains very small but if the pivot is the largest element we stack $n - 1$ problems to do later.

In Place

On average, height of the stack for Quicksort is $\Theta(\log n)$. Height of stack for Mergesort is $\lg n + O(1)$.

Definition: An algorithm is *in place* if it uses $O(1)$ extra variables and (on average) $O(\log n)$ extra index variables.

A more technical definition is; An algorithm is *in place* if it uses at most $O(1)$ extra variables and (on average) $O((\log n)^2)$ extra bits.

Stack

We can modify the Quicksort procedure so that stack has height at most $\lg n$.

```

procedure QUICKSORT( $A, p, r$ )
  if  $p < r$  then
     $q \leftarrow \text{PARTITION}(A, p, r)$ 
    if  $q \leq (p+r)/2$  then
      QUICKSORT( $A, p, q-1$ )
      QUICKSORT( $A, q+1, r$ )
    else
      QUICKSORT( $A, q+1, r$ )
      QUICKSORT( $A, p, q-1$ )
    end if
  end if
end procedure

```

Is Quicksort In Place?

Quicksort is in place, but it does not use a constant amount of extra space. Quicksort uses slightly more than a constant amount of extra space.

Choice of Pivots

It is risky to pivot on the last element because the last element could be the largest element. Some better ways could be;

- Pivot on middle element.
- Pivot on median of first, middle, and last elements.
- Pivot on random element.
- Pivot on median of three random elements.
- Pivot on median of five random elements. (Law of diminishing returns.)
- Randomly permute array before starting. (Equivalent to pivoting on random element.)

Partitioning

There are a variety of partition routines. The current edition of the textbook has a version that uses $n - 1$ comparisons, but the previous edition of the textbook has a version which uses n comparisons.

A Comparison of Sorting Algorithms

A.1 Temporal and Spatial Locality

Temporal

Temporal locality is dependent on how many variables you have and often you have to replace them.

```

sum ← 0
j ← 1
for i = 1 to 10 do
    sum ← sum + j
    j ← j + j
end for

```

The above program has good temporal locality if there are three registers, but bad temporal locality if there are only two registers (there are three unique variables in the program above so anything with less than three available registers will have bad temporal locality).

Spatial

If a particular storage location is referenced at a particular time, then it is likely that nearby memory locations will be referenced in the near future. In this case it is common to attempt to guess the size and shape of the area around the current reference for what it is worthwhile to prepare faster access for subsequent reference.

A.2 Comparison Table

Algorithm	Worst Comp.	Avg. Comp.	Worst Moves	Avg. Moves	Worst Exch.	Avg. Exch.	In Place	Spatial Locality
Bubble Sort	$n^2/2$	$n^2/2$			$n^2/2$	$n^2/2$	Yes	Yes
Insertion Sort	$n^2/2$	$n^2/4$	$n^2/2$	$n^2/4$			Yes	Yes
Selection Sort	$n^2/2$	$n^2/2$			n	n	Yes	Yes
Mergesort	$n \lg n$	$n \lg n$?	?			Y/N	Yes
Heapsort	" $n \lg n$ "	$n \lg n$	" $n \lg n$ "	$n \lg n$			Yes	No
Quicksort	$n^2/2$	$1.4n \lg n$?	?			Yes	Yes