

# Introduction to Algorithms



DEPARTMENT OF  
COMPUTER SCIENCE

Neelam Akula

Spring 2021

Last Updated: March 24, 2021

## Contents

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

|   |           |
|---|-----------|
| <b>9 Quicksort</b>                          | <b>23</b> |
| 9.1 Pseudocode . . . . .                    | 23        |
| 9.2 Analysis . . . . .                      | 23        |
| 9.3 Comments . . . . .                      | 27        |
| <b>A Comparison of Sorting Algorithms</b>   | <b>30</b> |
| 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 118059162071741130344 nodes. Level 71 has 236118324143482260688 nodes. Level 72 has 472236648286964521376 nodes. Level 73 has 944473296573929042752 nodes. Level 74 has 1888946593147858085504 nodes. Level 75 has 3777893186295716161008 nodes. Level 76 has 7555786372591432322016 nodes. Level 77 has 15111572745182864644032 nodes. Level 78 has 30223145490365729288064 nodes. Level 79 has 60446290980731458576128 nodes. Level 80 has 120892581961462917152256 nodes. Level 81 has 241785163922925834304512 nodes. Level 82 has 483570327845851668609024 nodes. Level 83 has 967140655691703337218048 nodes. Level 84 has 1934281311383406674436096 nodes. Level 85 has 3868562622766813348872192 nodes. Level 86 has 7737125245533626697744384 nodes. Level 87 has 15474250491067253395488768 nodes. Level 88 has 30948500982134506790977536 nodes. Level 89 has 61897001964269013581955072 nodes. Level 90 has 123794003928538027163910144 nodes. Level 91 has 247588007857076054327820288 nodes. Level 92 has 495176015714152108655640576 nodes. Level 93 has 990352031428304217311281152 nodes. Level 94 has 1980704062856608434622562304 nodes. Level 95 has 3961408125713216869245124608 nodes. Level 96 has 7922816251426433738490249216 nodes. Level 97 has 15845632502852867476980498432 nodes. Level 98 has 31691265005705734953960996864 nodes. Level 99 has 63382530011411469897921993728 nodes. Level 100 has 126765060022822939795843987456 nodes. Level 101 has 253530120045645879591687974912 nodes. Level 102 has 507060240091291759183375949824 nodes. Level 103 has 101412048018258351836675989648 nodes. Level 104 has 202824096036516703673351979296 nodes. Level 105 has 405648192073033407346703958592 nodes. Level 106 has 811296384146066814693407917184 nodes. Level 107 has 1622592768292133629386815834368 nodes. Level 108 has 3245185536584267258773631668736 nodes. Level 109 has 6490371073168534517547263337472 nodes. Level 110 has 12980742146337069035094526674944 nodes. Level 111 has 25961484292674138070189053349888 nodes. Level 112 has 51922968585348276140378106699776 nodes. Level 113 has 103845937170696552280756213399552 nodes. Level 114 has 207691874341393104561512426799104 nodes. Level 115 has 415383748682786209123024853598208 nodes. Level 116 has 830767497365572418246049707196416 nodes. Level 117 has 1661534994731144836492099414392832 nodes. Level 118 has 3323069989462289672984198828785664 nodes. Level 119 has 6646139978924579345968397657571328 nodes. Level 120 has 13292279957849158691936795315142656 nodes. Level 121 has 26584559915698317383873590630285312 nodes. Level 122 has 53169119831396634767747181260570624 nodes. Level 123 has 106338239662793269535494362521141248 nodes. Level 124 has 212676479325586539070988725042282496 nodes. Level 125 has 425352958651173078141977450084564992 nodes. Level 126 has 850705917302346156283954900169129984 nodes. Level 127 has 1701411834604692312567909800338259968 nodes. Level 128 has 3402823669209384625135819600676519936 nodes. Level 129 has 6805647338418769250271639201353039872 nodes. Level 130 has 13611294676837538500543278402706079744 nodes. Level 131 has 27222589353675077001086556805412159488 nodes. Level 132 has 54445178707350154002173113610824318976 nodes. Level 133 has 10889035741470030800434622722164863752 nodes. Level 134 has 21778071482940061600869245444329727504 nodes. Level 135 has 43556142965880123201738490888659455008 nodes. Level 136 has 87112285931760246403476981777318900016 nodes. Level 137 has 174224571863520492806953963554637800032 nodes. Level 138 has 348449143727040985613907927109275600064 nodes. Level 139 has 696898287454081971227815854218551200128 nodes. Level 140 has 139379657490816394245563170843710400256 nodes. Level 141 has 278759314981632788491126341687420800512 nodes. Level 142 has 557518629963265576982252683374841601024 nodes. Level 143 has 1115037259926531153964505366749683202048 nodes. Level 144 has 2230074519853062307929010733499366404096 nodes. Level 145 has 4460149039706124615858021466998732808192 nodes. Level 146 has 8920298079412249231716042933997465616384 nodes. Level 147 has 17840596158824498463432085867994931232768 nodes. Level 148 has 35681192317648996926864171735989862465536 nodes. Level 149 has 71362384635297993853728343471979724931072 nodes. Level 150 has 142724769270595987707456686943959498621544 nodes. Level 151 has 285449538541191975414913373887918997243088 nodes. Level 152 has 570898577082383950829826747775837994486176 nodes. Level 153 has 1141797154164767901659653495551675988972352 nodes. Level 154 has 2283594308329535803319306981103351977944704 nodes. Level 155 has 4567188616659071606638613962206703955889408 nodes. Level 156 has 9134377233318143213277227924413407911778816 nodes. Level 157 has 18268754466636286426554455848826815823557632 nodes. Level 158 has 36537508933272572853108911697653631647115264 nodes. Level 159 has 73075017866545145706217823395307263294230528 nodes. Level 160 has 146150035733090291412435646785614526588461056 nodes. Level 161 has 292300071466180582824871293571229053176922112 nodes. Level 162 has 584600142932361165649742587142458106353844224 nodes. Level 163 has 116920028586472233129548517428491621270768448 nodes. Level 164 has 233840057172944466259097034856983242541536896 nodes. Level 165 has 467680114345888932518194069713966485083073792 nodes. Level 166 has 935360228691777865036388139427932970166147584 nodes. Level 167 has 1870720457383555730072776278855865940332295168 nodes. Level 168 has 3741440914767111460145552557711731880664590336 nodes. Level 169 has 7482881829534222920291105115423463761329180672 nodes. Level 170 has 14965763659068445840582210230846927522658361344 nodes. Level 171 has 29931527318136891681164420461693855045316722688 nodes. Level 172 has 59863054636273783362328840923387710090633445376 nodes. Level 173 has 11972610927254756672465768184677542018126688656 nodes. Level 174 has 23945221854509513344931536369355084036253377312 nodes. Level 175 has 47890443709018526689863072738710168072506754624 nodes. Level 176 has 95780887418037053379726145477420336145013509248 nodes. Level 177 has 191561774836074106759452290954840672290027018496 nodes. Level 178 has 383123549672148213518904581909681344580054036992 nodes. Level 179 has 766247099344296427037809163819362689600108073984 nodes. Level 180 has 1532494198688592854075618327638725379200216147968 nodes. Level 181 has 3064988397377185708151236655277450758400432295936 nodes. Level 182 has 6129976794754371416302473310554901516800864591872 nodes. Level 183 has 12259953589508742832604946621109803033601729183744 nodes. Level 184 has 24519907179017485665209893242219606067203458367488 nodes. Level 185 has 49039814358034971330419786484439212134406916734976 nodes. Level 186 has 98079628716069942660839572968878424268813833469952 nodes. Level 187 has 196159257432139885321679145937756848537627666939808 nodes. Level 188 has 392318514864279770643358291875513697075255333879616 nodes. Level 189 has 784637029728559541286716583751027394150510667758232 nodes. Level 190 has 1569274059457119082573433167502054788301021335516464 nodes. Level 191 has 3138548118914238165146866335004109576602042671032928 nodes. Level 192 has 6277096237828476330293732670008219153204085342065856 nodes. Level 193 has 12554192475656952660587465340016438306408170684131712 nodes. Level 194 has 25108384951313905321174930680032876612816341368263424 nodes. Level 195 has 50216769852627810642349861360065753225632682736526848 nodes. Level 196 has 100433539705255621284697722720131506451265365473053696 nodes. Level 197 has 200867079410511242569395445440263012902530730946107392 nodes. Level 198 has 401734158821022485138790890880526025805061461892214784 nodes. Level 199 has 803468317642044970277581781761052051610122923784429568 nodes. Level 200 has 1606936635284089940555163563522104103220245847568859136 nodes. Level 201 has 3213873270568179881110327127044208206440491695137718272 nodes. Level 202 has 6427746541136359762220654254088416412880983390275436544 nodes. Level 203 has 12855493082272719524441308508176832825761966780548873088 nodes. Level 204 has 25710986164545439048882617016353665655523933561097746176 nodes. Level 205 has 51421972329090878097765234032707331311047867122195492352 nodes. Level 206 has 102843944658181756195530468065414662622095734244390984704 nodes. Level 207 has 205687889316363512391060936130829325244191468488781969408 nodes. Level 208 has 411375778632727024782121872261658650488382936975563938816 nodes. Level 209 has 822751557265454049564243744523317300976765873951127777632 nodes. Level 210 has 1645503114530908099128487489046634601953531747902255555264 nodes. Level 211 has 3291006229061816198256954978093269203907063495804511110528 nodes. Level 212 has 6582012458123632396513909956186538407814126981609022221056 nodes. Level 213 has 13164024916247264793027819912373076815628253963218044442112 nodes. Level 214 has 26328049832494529586055639824746153631256507926436088884224 nodes. Level 215 has 52656099664989059172111279649492307262513015852872177768448 nodes. Level 216 has 105312199329978118344222559298984614525026031705744355536896 nodes. Level 217 has 210624398659956236688445118597969229050052063411488711073792 nodes. Level 218 has 421248797319912473376890237195938458100104126822977422147584 nodes. Level 219 has 842497594639824946753780474385876916200208253645954844295168 nodes. Level 220 has 1684995189279649893507560948771753832400416507291909688590336 nodes. Level 221 has 3369990378559299787015121897543507664800833014583819377180672 nodes. Level 222 has 6739980757118599574030243795087015329601666029167638754361344 nodes. Level 223 has 13479961514237199148060487590174030659203332058335277508722688 nodes. Level 224 has 26959923028474398296120975180348061318406664116670555017445376 nodes. Level 225 has 53899846056948796592241950360696122636813328233341110034890752 nodes. Level 226 has 107799692113897993184483900721392245273626656466682220069781504 nodes. Level 227 has 215599384227795986368967800142784490547253312933364440139563008 nodes. Level 228 has 431198768455591972737935600285568981094506625866728880279126016 nodes. Level 229 has 862397536911183945475871200571137962189013251733457760558252032 nodes. Level 230 has 1724795073822367890951742401142279324378026503466915521116504064 nodes. Level 231 has 3449590147644735781903484802284558648756053006933831042233008128 nodes. Level 232 has 6899180295289471563806969604569117295112106013867662084466016256 nodes. Level 233 has 13798360590578943127613939209138234590224212027735324168932032512 nodes. Level 234 has 27596721181157886255227878418276469180448424055470648337864065024 nodes. Level 235 has 55193442362315772510455756836552938360896848110941296675728130048 nodes. Level 236 has 110386884724631545020911513673105876721793696221882593351456260096 nodes. Level 237 has 220773769449263090041823027346211753443587392443765186702912520192 nodes. Level 238 has 441547538898526180083646054692423506887174784887530373405825040384 nodes. Level 239 has 883095077797052360167292109384847013774349569774660746811650080768 nodes. Level 240 has 1766190155594104720334584218769694027548699139549321493623300161536 nodes. Level 241 has 3532380311188209440669168437539388055097398279098642987246600323072 nodes. Level 242 has 7064760622376418881338336875078776110194796558197285974493200646144 nodes. Level 243 has 14129521244752837762676673750157532203895931116394571948966401292288 nodes. Level 244 has 28259042489505675525353347500315064407781862232789143897932802584576 |             |

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
 

---

```

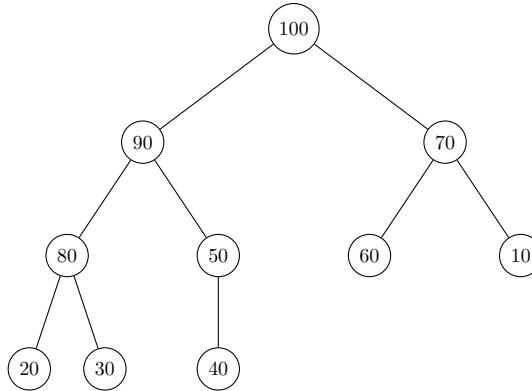
1: procedure HEAPSORT( $A, n$ )                                 $\triangleright A$  is a list and  $n$  is  $A$ 's size
2:   for  $r = \lfloor n/2 \rfloor$  down to 1 do                       $\triangleright$  Create Heap
3:     SIFT( $r, n$ )
4:   end for
5:
6:   for  $m = n$  down to 2 do                                 $\triangleright$  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

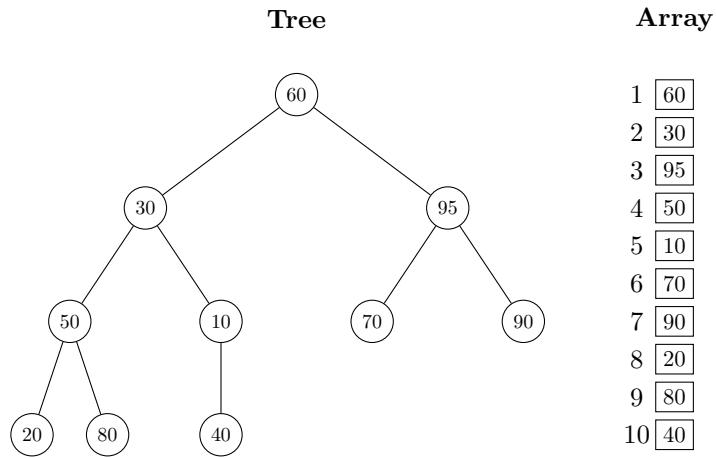


Figure 1: Use an array to implement a tree.  
 Node has index  $i$ , Left child is  $2i$ , Right child is  $2i + 1$ , and Parent is  $\lfloor \frac{i}{2} \rfloor$ .

### 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  $\alpha$ . The time to add two  $n$ -digit numbers is then  $\alpha n$ .

### 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.

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

#### 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.

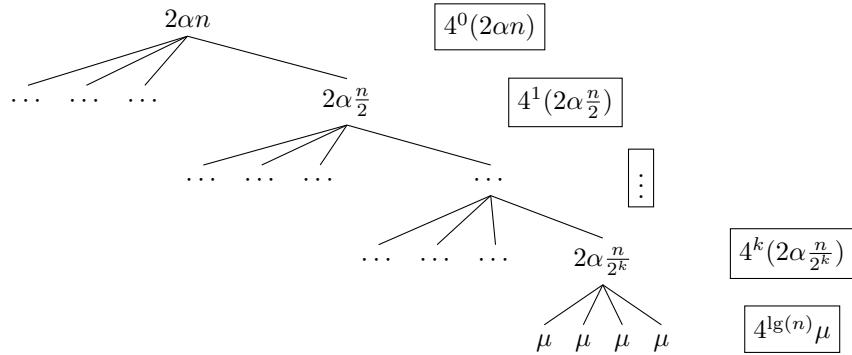
$$ad \times bc = ac + (ad + bc) + bd$$

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  $\alpha n$  time and  $ac + bd$  is also  $\alpha n$  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  $\mu$ . 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
 

---

```

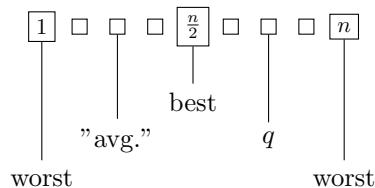
1: procedure QUICKSORT( $A, p, r$ )
2:   if  $p < r$  then
3:      $q \leftarrow \text{PARTITION}(A, p, r)$ 
4:     QUICKSORT( $A, p, q - 1$ )
5:     QUICKSORT( $A, q + 1, r$ )
6:   end if
7: end procedure

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              |