① $\mathbb{P}(X = i) = \binom{n}{i} p^i (1-p)^{n-i}$

- $\binom{n}{i}$: # of ways to pick i trials
- p^i: prob that these i trials are successes
- $(1-p)^{n-i}$: prob that the other trials failed

So $\mathbb{E}[X] = \sum_{i=0}^{n} i \cdot \binom{n}{i} p^i (1-p)^{n-i} = \sum_{i=1}^{n} n \binom{n-1}{i-1} p^i (1-p)^{n-i}$

$= np \sum_{i=1}^{n} \binom{n-1}{p-1} p^{i-1} (1-p)^{n-i} = np\,(p + [1-p])^{n-1} = np.$

② Let $X_i = \begin{cases} 0 & \text{if } i\text{th trial failed} \\ 1 & \text{if } i\text{th trial succeeded} \end{cases}$

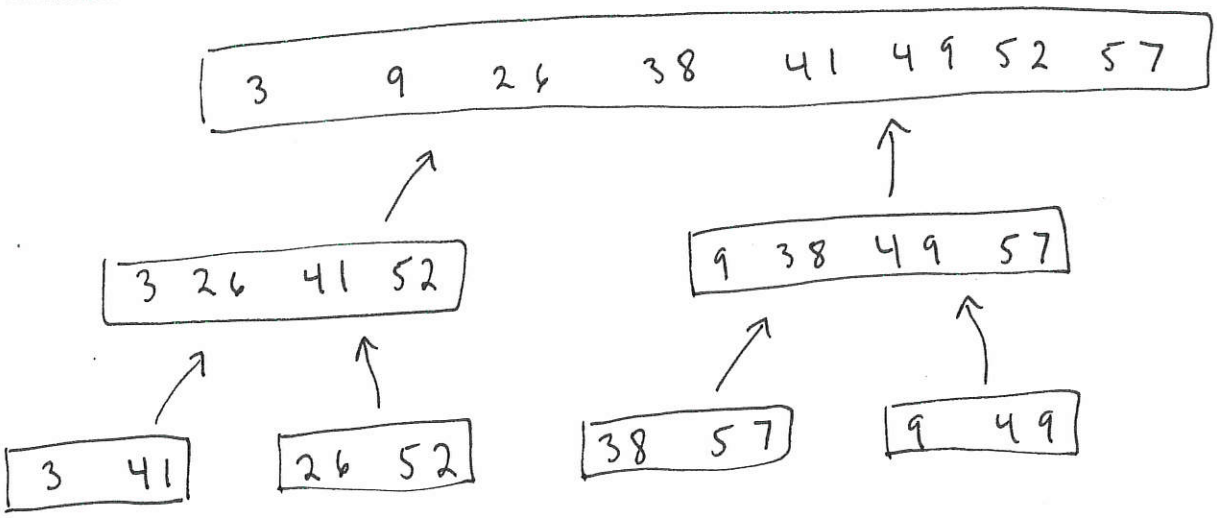
Then $X = X_1 + X_2 + \cdots + X_n$

So by linearity of expectation:

$$\mathbb{E}[X] = \mathbb{E}[X_1 + \cdots + X_n] = \sum_{i=1}^{n} \mathbb{P}(X_i = 1) = \sum_{i=1}^{n} p = np.$$

(3)

Ex 2.3-1



Ex 2.3-2

MERGE(A, p, q, r)

$n_1 = q - p + 1$

$n_2 = r - q$

let $L = [1 \dots n_1]$ and $R = [1 \dots n_2]$ be new arrays

for $i = 1$ to n_1

$L[i] = A[p + i - 1]$

for $j = 1$ to n_2

$R[j] = A[q + j]$

$i = 1$

$j = 1$

$k = p$

while $i < n_1 + 1$ and $j < n_2 + 1$

if $L[i] \leq R[j]$

$A[k] = L[i]$

$i = i + 1$

else $A[k] = R[j]$

$j = i + 1$

$k = k + 1$

end while loop

if $i = n_1 + 1$

for $j < n_2 + 1$

$A[k] = R[j]$

$j = j + 1$

$k = k + 1$

else for $i < n_1 + 1$

$A[k] = L[i]$

$i = i + 1$

$k = k + 1$

Ex 2.3 - 3

base case

$n = 2 \rightarrow T(n) = 2$

$n \lg n = 2 \cdot \lg 2 = 2$ ✓

induction step

assume $T(2^i) = 2^i \lg(2^i)$

then $T(2^{i+1}) = 2T(2^{i+1}/2) + 2^{i+1}$

$= 2T(2^i) + 2^{i+1} = 2^{i+1}\lg(2^i) + 2^{i+1}$

$= 2^{i+1}(\lg(2^i) + 1) = 2^{i+1}(\lg(2^i) + \lg(2))$

$= 2^{i+1}\lg(2^i \cdot 2) = 2^{i+1}\lg(2^{i+1})$ □

Ex 2.3 - 6

Yes, insertion-sort, in its worst case, starts w/ entry 2 and compares it to 1, then takes entry 3, compares it to 2 and then to 1, all the way to comparing the nth entry to all $n-1$ before it. This gives $\sum_{i=1}^{n-1} i$ comparisons. If we replace each i w/ at most $c \lg i$, as in binary search (since ~~pro~~ 2.3-5 says it is $\Theta(\lg n)$), then the new running time is

$$\sum_{i=1}^{n-1} c \lg i = c(\lg 1 + \cdots + \lg(n-1)) \leq c(\lg 1 + \cdots + \lg n)$$

$$\leq c(\lg n + \cdots + \lg n) = cn \lg n = \Theta(n \lg n).$$

(4) Prob 2-4

a. $2 > 1 \rightsquigarrow (1, 5)$ is an inversion

$3 > 1 \rightsquigarrow (2, 5)$ "

$8 > 6 \rightsquigarrow (3, 4)$ "

$8 > 1 \rightsquigarrow (3, 5)$ "

$6 > 1 \rightsquigarrow (4, 5)$ "

↑ these are their positions

b. in a set of n elements we can make at most $\binom{n}{2}$ comparisons. So there can be at most $\binom{n}{2}$ inversions.

the array $A = [n, n-1, \ldots, 2, 1]$ gives the max, since for any $1 \leq i < j \leq n$, $A[i] = n - i + 1$, $A[j] = n - j + 1$

and $i < j \Rightarrow n - i + 1 > n - j + 1$.

c. at each step we compare the ith element to each one before it until it isn't the lowest of the two. So the number of comparisons made is equal to the number of inversions it was a part of w/ lower positions plus one* So

$$\left[\begin{array}{c}\#\text{ of} \\ \text{inversions}\end{array}\right] \leq \left[\begin{array}{c}\text{Run time} \\ \text{of Insertion Sort} \\ \text{on } n\end{array}\right] \leq (n-1) + \left[\begin{array}{c}\#\text{ of} \\ \text{inversions}\end{array}\right]$$

* the lower list is already in order so stops after # of lower inversions *

* each of $2, \_\_, n$ gets compared to the lower list at least once *

* unless everything below it was inverted

(5) I don't have the slides for this question, but I found Bubblesort in the book. Hopefully, it's the same kind of thing:

```
BubbleSort (A)
    for i = 1 to A.length - 1
        for j = A.length down to i+1
            if A[j] < A[j-1]
                exchange A[j] with A[j-1].
```

the inside for loop makes $n - i$ comparisons and the outside loop goes from $i = 1$ to $n-1$.

So it has runtime $\sum_{i=1}^{n-1} (n-i) = \sum_{i=1}^{n-1} i = \frac{1}{2} n(n-1)$

$= \binom{n}{2}$

$= \Theta(n^2)$.

this is both the best and the worst case since nothing in the algorithm would ever stop it from making this many comparisons.

(6)

```
Insertion Sort (A)
    for j = 2 to A.length
        key = A[j]
        i = j - 1
        while i > 0 and A[i] > key
            A[i+1] = A[i]
            i = i - 1
        A[i+1] = key
```

In #3 Ex 2.3-6 of this homework we see that the worst case runtime of insertion sort is $\sum_{i=1}^{n-1} i$

in the event that the while loop above goes the full $j-1$ times possible for each $j = 2, \ldots, n$.

So worst case is $\frac{1}{2}n(n-1) = \Theta(n^2)$.

Best case happens when the while loop is as short as possible ~ so when $A[j-1] < key$ and there is only one comparison. This gives a runtime of $\sum_{j=2}^{n} 1 = n - 1 = \Theta(n)$.

It is easy to see that worst happens for $[n, n-1, \ldots, 1]$ and best happens for $[1, 2, \ldots, n]$.

from problem 4 we know that the runtime is more or less the same as the number of inversions in the array. So I'm going to say that the average runtime is going to be "about" the expected number of inversions in a random array of n elements, give or take a linear factor.

So Let $X = \left[\begin{array}{l}\#\text{ of} \\ \text{inversions} \\ \text{of an array} \\ \text{of } n \text{ numbers}\end{array}\right]$ and let $X_{(i,j)} = \begin{cases} 0 & \text{if } A[i] \leq A[j] \\ 1 & \text{if } A[j] < A[i] \end{cases}$

then $X = \sum_{i<j} X_{(i,j)}$

so $\mathbb{E}(X) = \sum_{i<j} \mathbb{E}(X_{(i,j)}) = \binom{n}{2} \mathbb{P}\left(A[j] < A[i] \text{ for some } i<j\right)$

$= \frac{1}{2}\binom{n}{2}$

since for any 2 numbers of the n there are 2 options if we need to place them in i or j — either in order or out of order. So $\mathbb{P}(A[i] < A[j]) = \frac{1}{2}$.

So I expect the runtime to be on average about $\frac{1}{2}\binom{n}{2} \pm n = \Theta(n^2)$.

(7) Let n be some integer, then these statements are certainly true since

$$\lfloor -n \rfloor = \lceil -n \rceil = -\lceil n \rceil = -\lfloor n \rfloor = -n.$$

Now suppose x is not an integer, then $\exists$ some integer n such that $n < x < n+1$

so $-n > -x > -n-1$

and

$$\lfloor -x \rfloor = -n-1$$

$$\lceil -x \rceil = -n$$

$$-\lfloor x \rfloor = -n$$

$$-\lceil x \rceil = -n-1$$

Hence, $\lfloor -x \rfloor = -\lceil x \rceil$ and $\lceil -x \rceil = -\lfloor x \rfloor$ $\square$

6.1-1

the height of a heap is the maximum # of edges from the root down to a leaf
- so, one less than the number of levels

If we number the levels $0, 1, 2, \ldots, h$ then we can see that at level i the first index is 2^i and the last index is $2^{i+1} - 1$.

So a heap of height h has at least 2^h and at most $2^{h+1} - 1$ elements.

6.1-2

any integer n is in between two successive powers of 2: $2^i \leq n < 2^{i+1}$

So an n-element heap must have height $i = \lfloor \lg n \rfloor$.

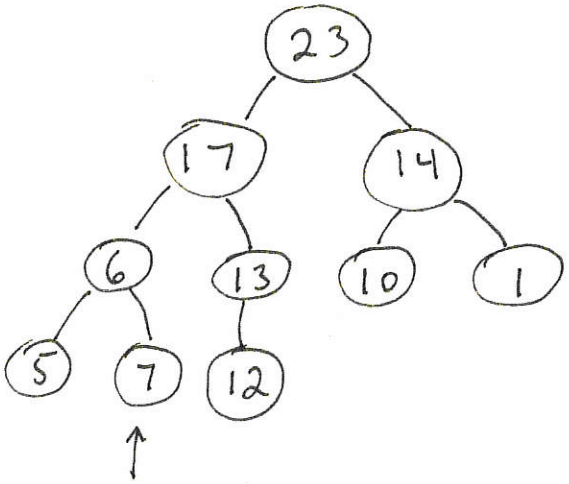
6.1-4

max-heap only requires that parents be larger (or the same as) their children. So if all elements are distinct, then the smallest element must be on a childless node. If the lowest level of the tree is full then that's where the smallest is. Otherwise, it could be on any of the lowest level or any of the childless nodes on the second lowest level.

6.1-5 Yes, child nodes always have index higher than the parent so if the array is sorted then the child element is greater than or equal to the parent. So it is a min-heap.

6.1-6 $\langle 23, 17, 14, 6, 13, 10, 1, 5, 7, 12 \rangle$

$\Rightarrow$



no, not a max-heap since $7 > 6$.

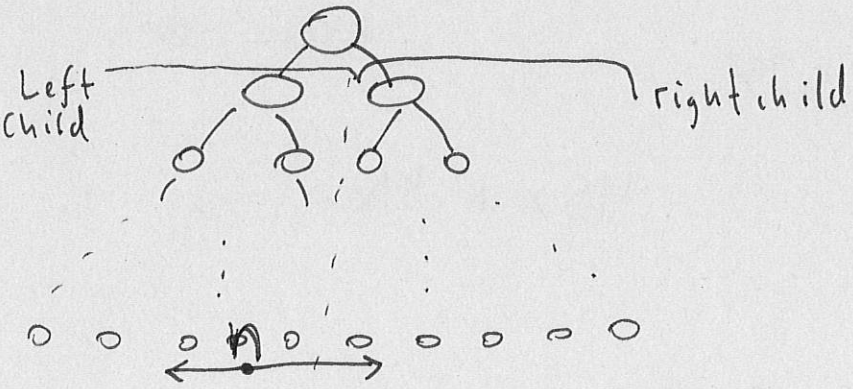
6.1-7 If $2^i \leq n < 2^{i+1}$, then the leaves are at indices $2^i, 2^i+1, \ldots, n$ and at the Parents of $n+1, n+2, \ldots, 2^{i+1}-1$ unless $\text{Parent}(n+1) = \text{Parent}(n)$ in which case just the parents of $n+2, \ldots, 2^{i+1}-1$.

$\text{Parent}(n+1) = \text{Parent}(n) \iff n$ is even

So if n is even then the leaves are at $\left\lfloor \frac{n+2}{2} \right\rfloor = \lfloor n/2 \rfloor + 1, n/2 + 2, \ldots, n$

if n is odd then the leaves are at $\left\lfloor \frac{n+1}{2} \right\rfloor = \lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, \ldots, n$.

(9) first note that since the bottom level of the left child is always filled in before the bottom level of the right child, then it is always bigger. As a percentage of n it decreases the larger n is past halfway across the bottom level.



n can be any of these
but Left always has at least
as much as Right

So if Left Size $= \alpha n$ then we want to maximize α

w/ some $n = 2^i, 2^i+1, \ldots, 2^i + (2^{i-1} - 1)$

Left has total $n - 1 - \text{RightSize} = n - 1 - (2^{i-1} - 1)$

So maximize $$\alpha = \frac{n - 1 - (2^{i-1} - 1)}{n} = 1 - \frac{2^{i-1}}{n}$$

so n must be as large as possible on the interval.

So $n = 2^i + 2^{i-1} - 1 = 2^{i-1}(2+1) - 1 = 3 \cdot 2^{i-1} - 1$.

$$\alpha = 1 - \frac{2^{i-1}}{3 \cdot 2^{i-1} - 1} < 1 - \frac{1}{3} = 2/3. \quad \text{So } |\text{Left child}| < 2/3\, n.$$

Therefore, $T(n) = T(\frac{2}{3}n) + c$

so by master theorem since $\log_{3/2} 1 = 0$

and $c = \Theta(n^0)$ so

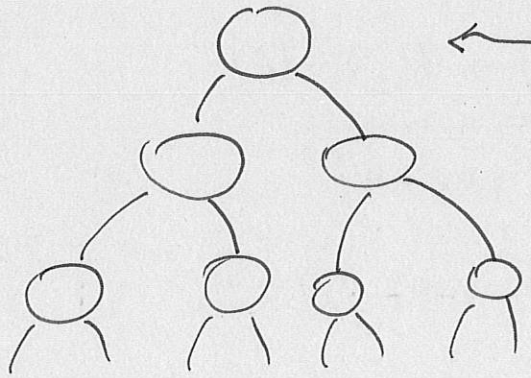
$T(n) = \Theta(n^0 \lg n) = \Theta(\lg n)$. in worst case.

$\Rightarrow T(n) = O(\lg n)$ in general.

(10) Not sure what argument was used in class, but here's one:

Max-heapify takes at most $O(\lg n)$ on a heap of n elements. Since the root is at height $h = \lfloor \lg n \rfloor$, then max-heapify done at index i of an n-element heap takes at most $O(h_i)$ where h_i is the height of index i.

So:



$\leftarrow O(\lg n)$

$\leftarrow 2O(\lg n - 1)$

$\leftarrow 4O(\lg n - 2)$

$\vdots$

$< 2^{\lg n - 1} O(\lg n - (\lg n - 1))$

$\lfloor n/2 \rfloor$

$$\text{Sum} = \sum_{i=0}^{\lg n - 1} 2^i O(\lg n - i) \leq c\left(\sum_{i=0}^{\lg n - 1} 2^i \lg n - \sum_{i=0}^{\lg n - 1} i 2^i\right)$$

$$= c\left[\lg n\left(1+2+2^2+\cdots+2^{\lg n-1}\right) - 2\sum_{i=1}^{\lg n-1} i\cdot 2^{i-1}\right]$$

$$= c\left[\lg n\left(\frac{2^{\lg n}-1}{2-1}\right) - 2\,\frac{d}{dx}\left(\sum_{i=1}^{\lg n-1} x^i\right)_{x=2}\right]$$

$$= c\left[(n-1)\lg n - 2\,\frac{d}{dx}\left(\frac{x^{\lg n}-1}{x-1} - 1\right)_{x=2}\right]$$

$$= cn\lg n - c\lg n - 2c\left(\frac{\lg n\, x^{\lg n-1}(x-1) - x^{\lg n}+1}{(x-1)^2}\right)_{x=2}$$

$$= cn\lg n - c\lg n - 2c(\lg n)(n/2) + 2nc - 2c$$

$$= cn\lg n - c\lg n - cn\lg n + 2nc - 2c$$

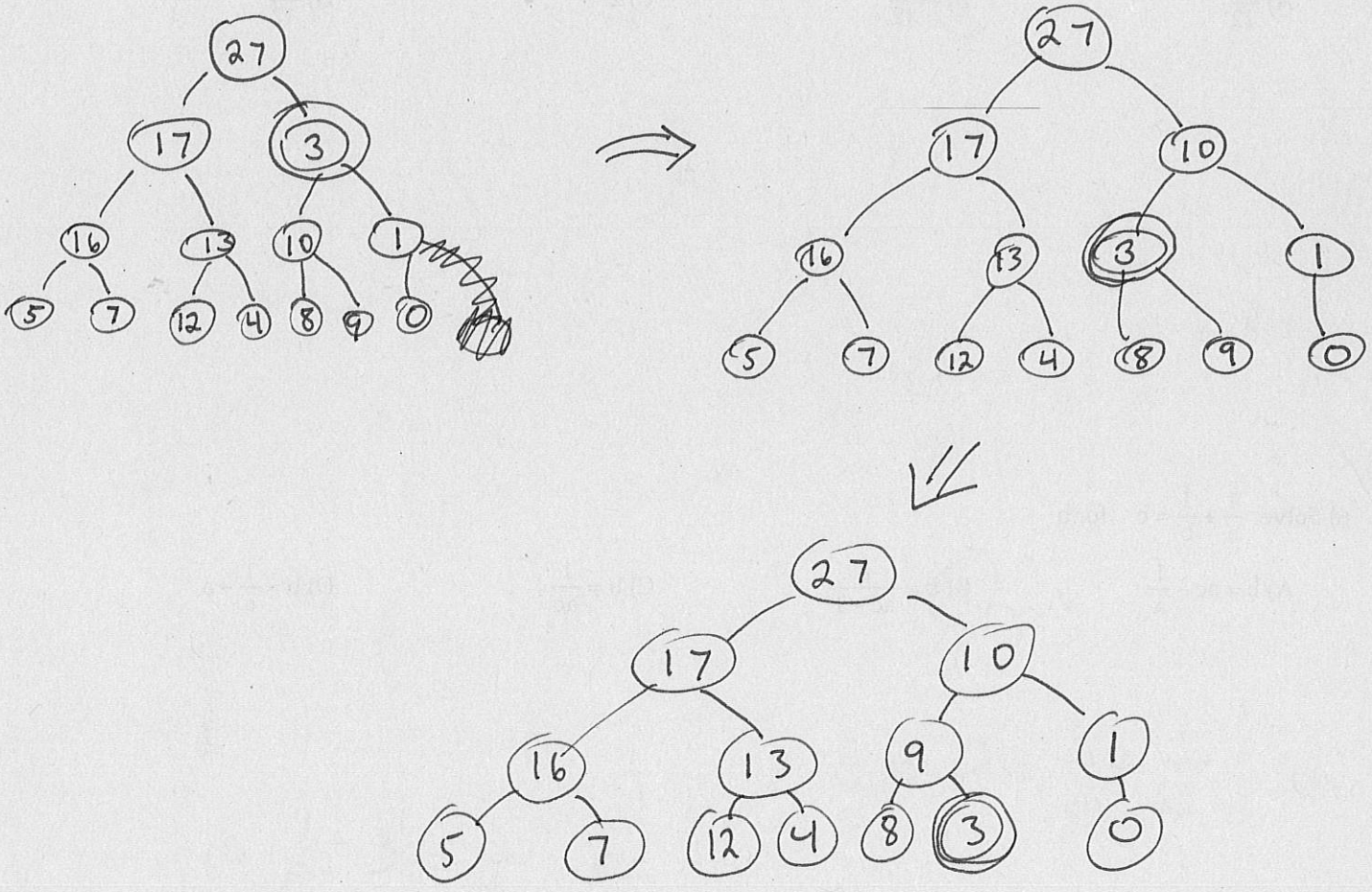
$$= 2cn - c\lg n - 2c \le 2cn = O(n)$$

□

11

6.2-1 MAX-HEAPIFY(A, 3)

$A = \langle 27, 17, 3, 16, 13, 10, 1, 5, 7, 12, 4, 8, 9, 0 \rangle$



6.2-3 Nothing is exchanged.

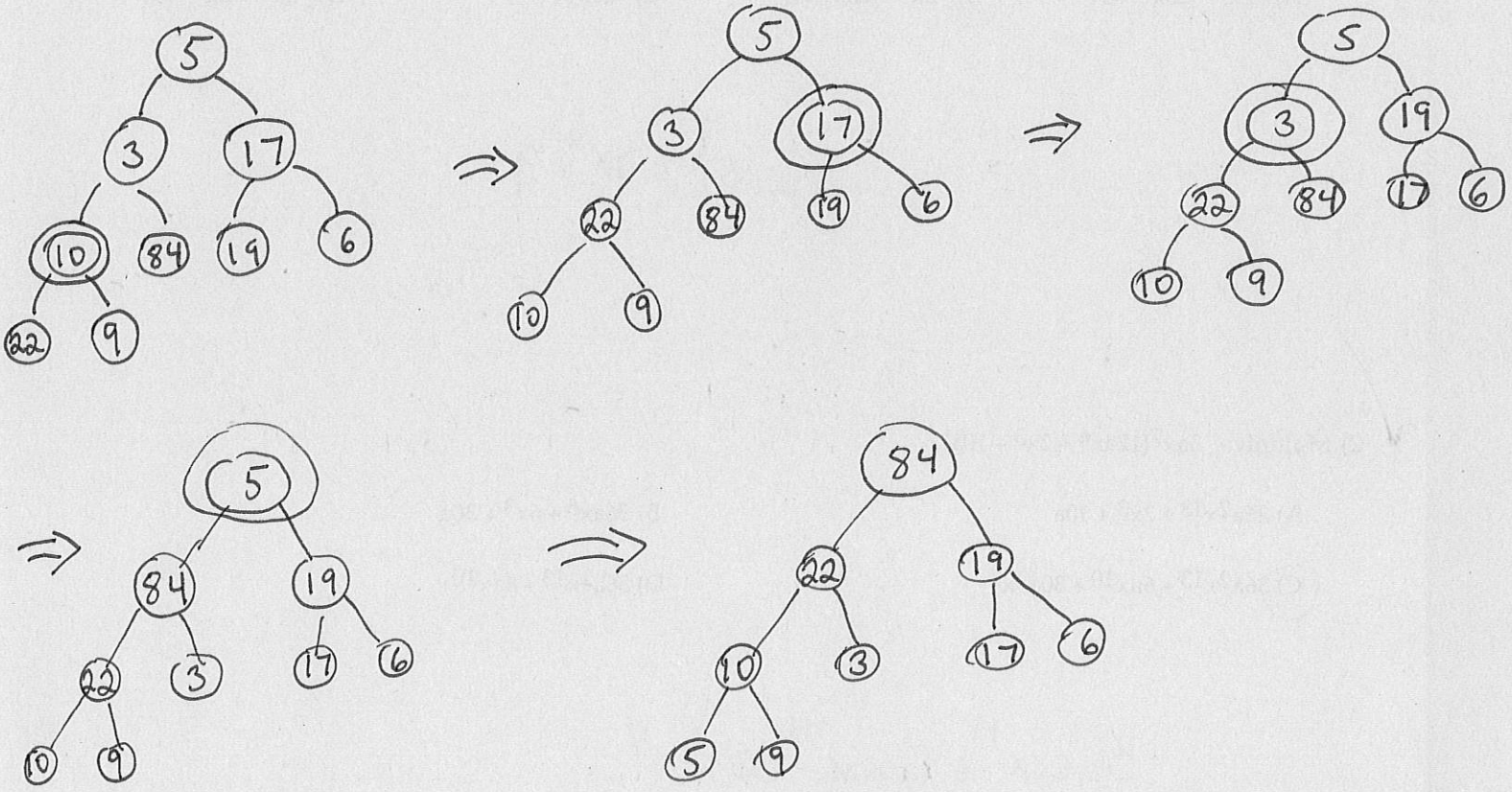
6.2-4 Then $2i > A.heapsize$ so no element of the heap is a child of the element at index i. So the algorithm changes nothing.

6.2-6 I showed in problem 9 that $T(n) = \Theta(\lg n)$ in the worst case. So $T(n) = \Omega(\lg n)$.

12

6.3-1

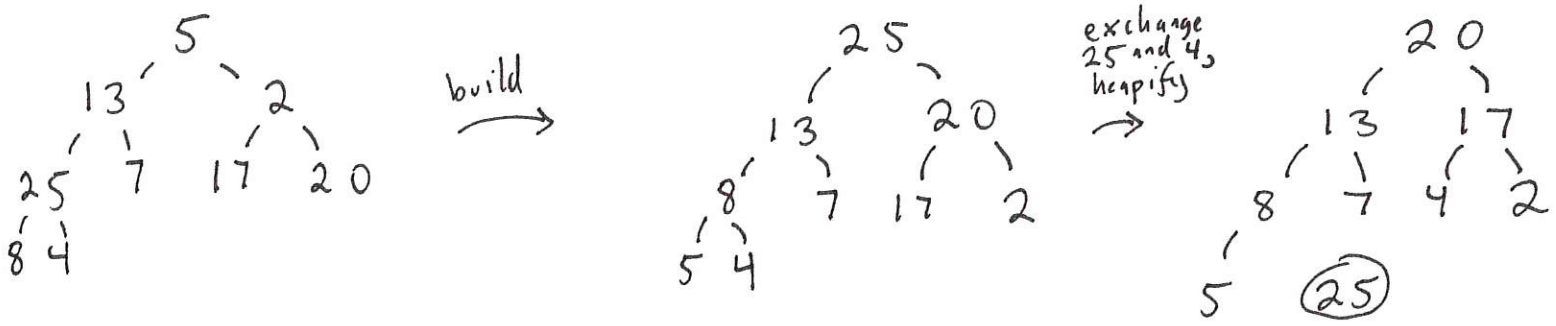
Max Build Heap (A) for $A = \langle 5, 3, 17, 10, 84, 19, 6, 22, 9 \rangle$

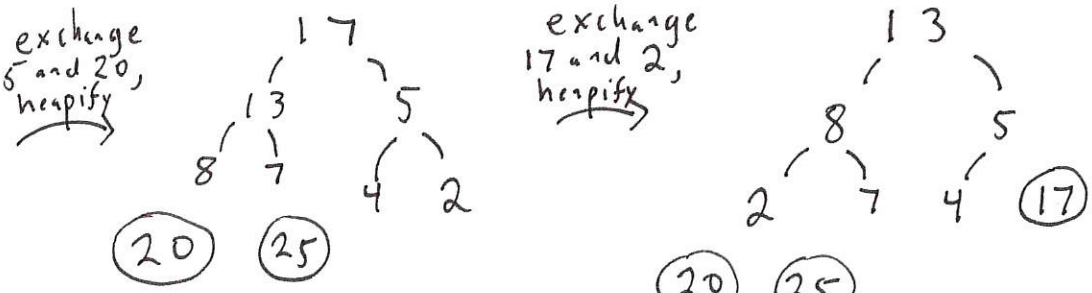


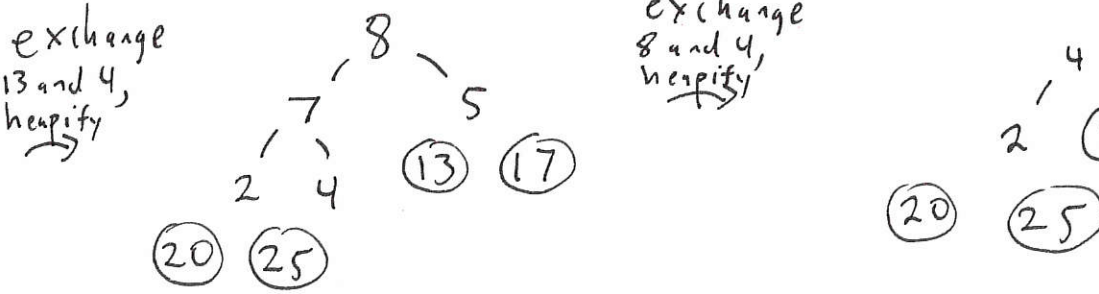
6.3-3 The # of nodes at each level starting from the root and moving down is $1, 2, 4, \ldots, 2^i, \ldots, \leq 2^h$.

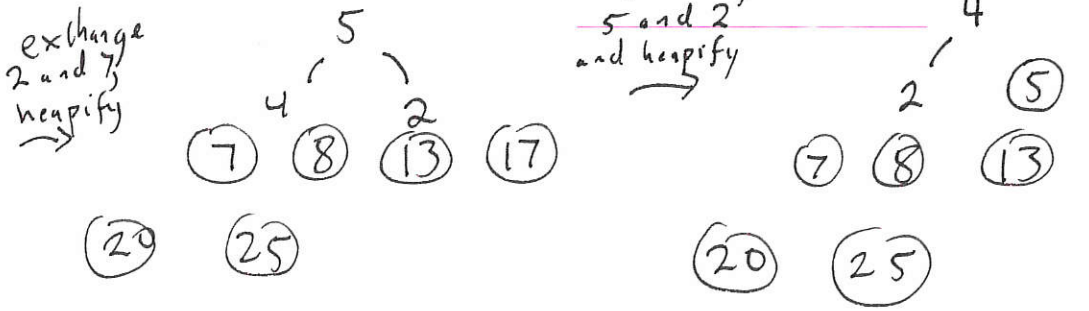
So each level has (roughly) half of the level below it. and since $h \approx \lg n - 1$ then the bottom has $\sim n/2$ and each successive level has half the one before so $n/2^{h_i+1}$ where h_i is the height of the ~~index of the~~ node at index i.

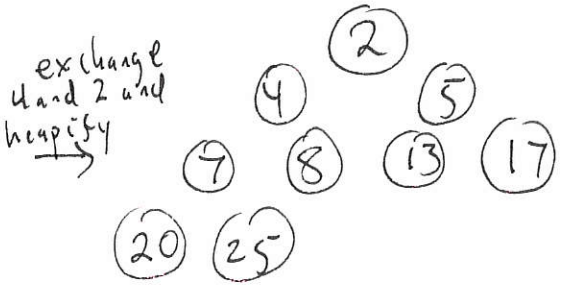
(13) 6.4-1 HeapSort(A) on $A = \langle 5, 13, 2, 25, 7, 17, 20, 8, 4 \rangle$







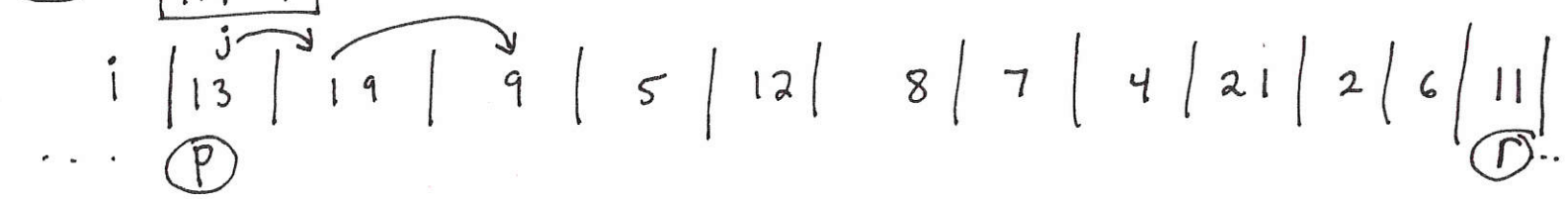




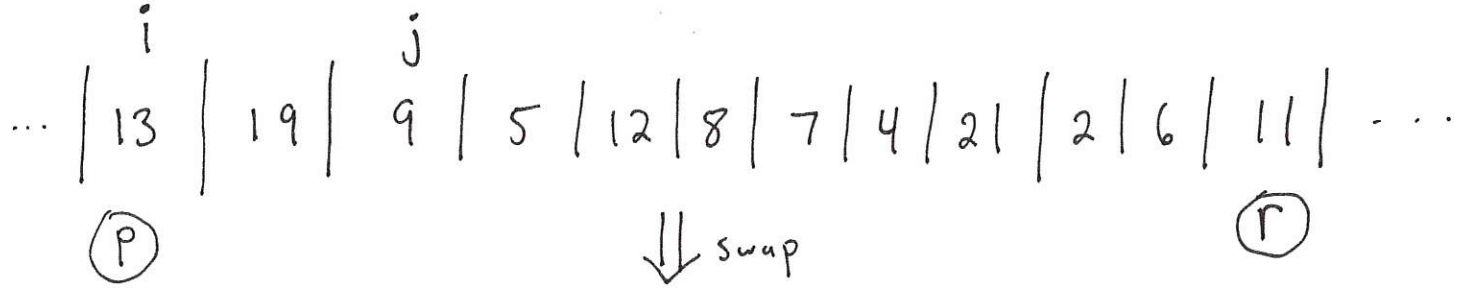
6.4-3 and 6.4-4

I think that the issue here is that no matter what array you start with it will be put into a max heap (always linear time) and then each of the n-1 calls to max heapify will take the full time since the algorithm puts an element from the bottom level on top of a heap that is max otherwise. So this element must go all the way or "almost" all of the way down the tree. So heapsort is $\Theta(n \lg n)$.

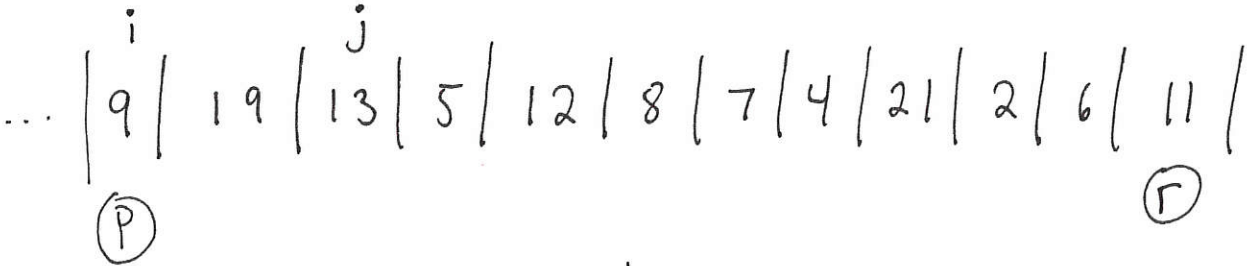
(14) 7.1-1



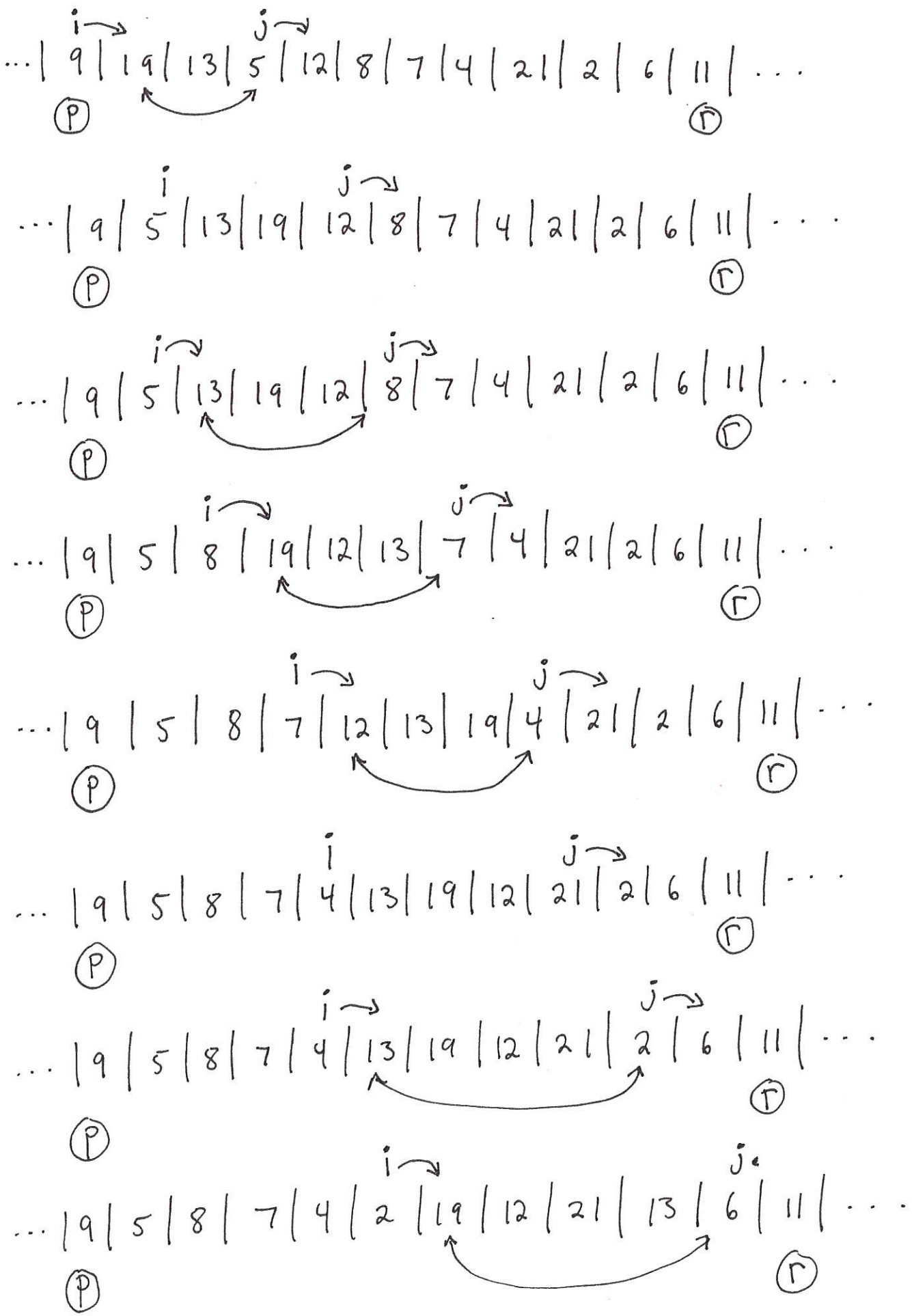
13, 19 > 11
so first 2 steps just move j up 2 spots
then $9 \leq 11$ so i moves up and we swap:

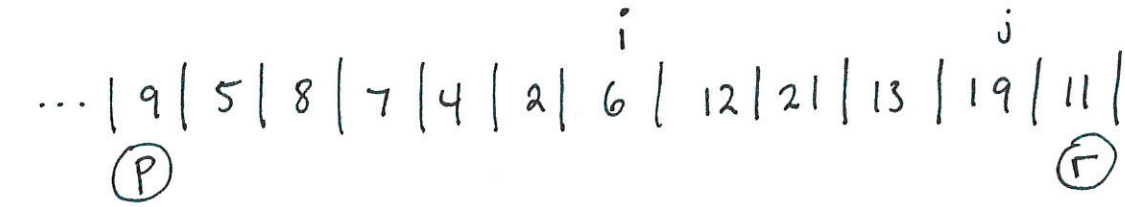


swap



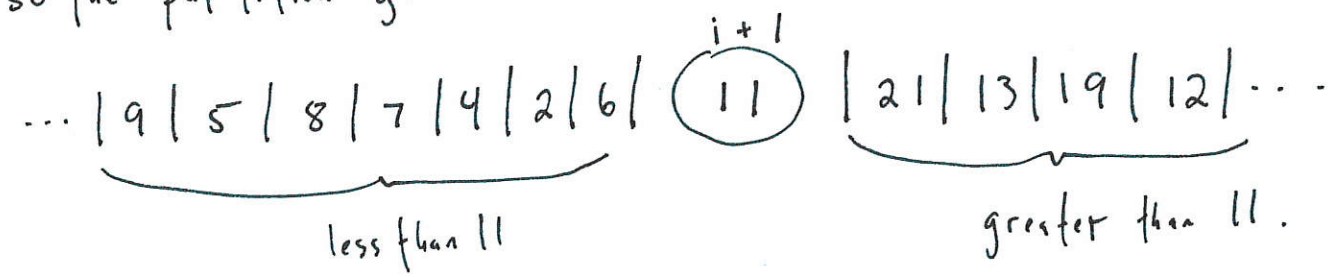
and so on:


i
j
... | 9 | 19 | 13 | 5 | 12 | 8 | 7 | 4 | 21 | 2 | 6 | 11 | ...
p
r
i
j
... | 9 | 5 | 13 | 19 | 12 | 8 | 7 | 4 | 21 | 2 | 6 | 11 | ...
p
r
i
j
... | 9 | 5 | 13 | 19 | 12 | 8 | 7 | 4 | 21 | 2 | 6 | 11 | ...
p
r
i
j
... | 9 | 5 | 8 | 19 | 12 | 13 | 7 | 4 | 21 | 2 | 6 | 11 | ...
p
r
i
j
... | 9 | 5 | 8 | 7 | 12 | 13 | 19 | 4 | 21 | 2 | 6 | 11 | ...
p
r
i
j
... | 9 | 5 | 8 | 7 | 4 | 13 | 19 | 12 | 21 | 2 | 6 | 11 | ...
p
r
i
j
... | 9 | 5 | 8 | 7 | 4 | 13 | 19 | 12 | 21 | 2 | 6 | 11 | ...
p
r
i
j
... | 9 | 5 | 8 | 7 | 4 | 2 | 19 | 12 | 21 | 13 | 6 | 11 | ...
p
r



now we exchange i+1 and r and return i+1

so the partition gives



7.1-2

if they are all the same, then $A[j] \leq A[r]$ for each $j = p, \ldots, r-1$. So i starts at $p-1$ and increases by one $r-p$ times so $i = r-1$ at the end which means $q = i+1 = r$.

there are many ways the program could be modified to do this. here's one way:

change the for loop to:

```
s = 0
for j = p to r-1
    d = x - A[j]
    s = s + d
    if 0 ≤ d
        i = i + 1
        exchange A[i] with A[j]
exchange A[i+1] with A[r]
if s = 0
    return ⌊(r+p)/2⌋
else
    return i + 1
```

7.1-3

the for loop runs $n-1$ times and does a constant number of things each time regardless of the particular array. So best and worst times are constant times $n-1$: $\Theta(n)$.

(15) 7-3

a) suppose we have n distinct elements to put into an array of size n

the probability that a particular element is chosen as the pivot is the probability that this element is last in the array. there are $n!$ possible arrays and $(n-1)!$ of them have this element at the end so the probability is $\frac{(n-1)!}{n!} = 1/n$.

if $X_i := \begin{cases} 1 & \text{if } i\text{th smallest is the pivot} \\ 0 & \text{otherwise} \end{cases}$

then $\mathbb{E}[X_i] = \mathbb{P}(X_i = 1) \cdot 1 + \mathbb{P}(X_i = 0) \cdot 0$

$= 1/n$

b) if the q^{th} smallest element is the pivot then

$$T(n) = T(q-1) + T(n-q) + \Theta(n)$$

$T(q-1)$: quicksort less than q

$T(n-q)$: quicksort more than q

$\Theta(n)$: time it took to put elements above or below q

Therefore, $T(n) = \sum_{q=1}^{n} X_q \left(T(q-1) + T(n-q) + \Theta(n) \right)$

~~since if $q = i$ then X_i~~

since if i is the pivot then $X_i = 1$ and $X_{q \neq i} = 0$.

So $\mathbb{E}[T(n)] = \mathbb{E}\left[\sum_{q=1}^{n} X_q \left(T(q-1) + T(n-q) + \Theta(n) \right) \right]$

c) since $\mathbb{E}\left[X_q \left(T(n-q) + T(q-1) + \Theta(n) \right) \right]$

$= \mathbb{P}(q \text{ is the pivot}) \left(T(n-q) + T(q-1) + \Theta(n) \right)$

$+ \mathbb{P}(q \text{ is not the pivot}) \cdot 0$

then $\mathbb{E}[T(n)] = \sum_{q=1}^{n} \frac{1}{n} \mathbb{E}\left(T(n-q) + T(q-1) + \Theta(n) \right)$

$= \frac{1}{n} \mathbb{E}\left(\sum_{q=1}^{n} T(n-q) + \sum_{q=1}^{n} T(q-1) + \sum_{q=1}^{n} \Theta(n) \right)$

$= \frac{1}{n} \mathbb{E}\left(T(n-1) + \cdots + T(0) + T(0) + \cdots + T(n-1) + \Theta(n^2) \right)$

$= \frac{1}{n} \mathbb{E}\left(\sum_{q=2}^{n-1} 2\,T(q) + \Theta(n) \right) = \frac{2}{n} \sum_{q=2}^{n-1} \mathbb{E}(T(q)) + \Theta(n)$

$(T(0) = 0)$ $(T(1) = c)$

d) I couldn't get the hint to work so I used the integral upper bound since $k \lg k$ is monotone increasing:

$$\sum_{k=2}^{n-1} k \lg k \leq \int_2^n x \lg x \, dx = \frac{1}{2} x^2 \lg x \Big|_2^n - \int_2^n \frac{1}{2} x \, dx$$

(~~partial~~ by parts $u = \lg x \quad v = \frac{1}{2}x^2$

$du = \frac{1}{x} dx \quad dv = x \, dx$

$$= \frac{1}{2} x^2 \lg x - \frac{1}{4} x^2 \Big|_2^n$$

$$= \frac{1}{2} n^2 \lg n - \frac{1}{4} n^2 - 2 + 1$$

$$= \frac{1}{2} n^2 \lg n - \frac{1}{4} n^2 - 1 \leq \frac{1}{2} n^2 \lg n - \frac{1}{8} n^2$$

since

$$-\frac{1}{4} n^2 - 1 \leq -\frac{1}{8} n^2$$

$$\updownarrow$$

$$2n^2 + 8 \geq n^2$$

$$\updownarrow$$

$$n^2 \geq -8$$

always.

e) $$\mathbb{E}[T(n)] = \frac{2}{n}\sum_{q=2}^{n-1}\mathbb{E}[T(q)] + \Theta(n)$$

$$\overset{\text{induction step}}{=} \frac{2}{n}\sum_{q=2}^{n-1}\Theta(q\lg q) + \Theta(n)$$

$$c_2\sum_{q=2}^{n-1} q\lg q \le \sum_{q=2}^{n-1}\Theta(q\lg q) \le c_1\sum_{q=2}^{n-1} q\lg q$$

$$c_2\left(\tfrac{1}{2}n^2\lg n - \tfrac{1}{8}n^2\right) \le \sum_{q=2}^{n-1}\Theta(q\lg q) \le c_1\left(\tfrac{1}{2}n^2\lg n - \tfrac{1}{8}n^2\right)$$

$$\text{so } \sum_{q=2}^{n-1}\Theta(q\lg q) = \Theta(n^2\lg n)$$

$$\Rightarrow \mathbb{E}[T(n)] = \frac{2}{n}\Theta(n^2\lg n) + \Theta(n)$$

$$= \Theta(n\lg n).$$

(16) a) a presorted array gives the highest value as the pivot so the recurrence looks like

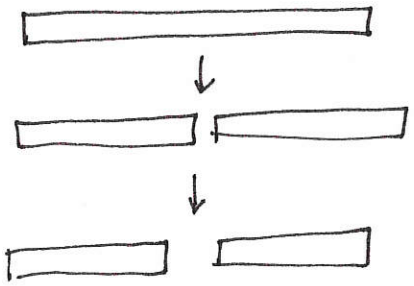
$$T(n) = T(n-1) + \Theta(n) = \Theta(n^2)$$

since $T(n) = \sum_{i=1}^{n} \Theta(i) \approx \Theta(\frac{1}{2}n^2)$ →

b) Same as before, $O(n^2)$, since it gives lowest value as the pivot.

c) Same, $O(n^2)$, since in #14 we found that this would give us the last index as pivot.

d)



good case gives pivot about $n/2$

bad case gives pivot $n/2 - 1$ for each

So $T(n) = 2T(n/2) + 2T(n/2 - 1) + \Theta(n)$

$\approx 4T(n/2) + \Theta(n)$

Master's Thm: $\log_2 4 = 2$ and $\Theta(n) = O(n^{2-\varepsilon})$ for any $0 < \varepsilon < 1$

So $T(n) = \Theta(n^2)$.