Intro. to Algorithm Assignment 2 2018L257 최찬혁.

1. (1 points) Explain that the running time of the PARTITION procedure of quicksort on a subarray of size n is $\Theta(n)$.

PARTITION(A, p, r) : running time analysis.

$x = A[r]$ ... 1

$i = p-1$ ... 1

for $j = p$ to $r-1$ ... $r-p$

if $A[p] \le x$ then ... $r-p$

$i = i+1$ ... $r-p$

exchange $A[i]$ with $A[j]$ ... $r-p$

exchange $A[i+1]$ with $A[r]$ ... 1

return $i+1$ ... 1

$\therefore$ total : $4(r-p+1)$

Since a size of a subarray is " n ", i.e., $n = r-p+1$, where r and p is a constant, the running time of the PARTITION should be $\underline{\underline{\Theta(n)}}$.

Because the running time in this function is mainly affected by <u>FOR</u> loop, and it has only one <u>FOR</u> loop.

2. (2 points) Chebyshev's inequality says that the probability that a random variable is more than k standard deviations away from the mean is less than $1/k^2$. For $N = 1{,}000{,}000$, use Chebyshev's inequality to bound the probability that the number of compares used by quicksort is more than 100 million.

Hint: Quicksort uses $2N \ln N$ (ln is the natural logarithm) compares on the average case (mean) to sort N keys, and the standard deviation of the number of compares is $0.65N$.

Let X be the number of compares used by quicksort :

$$E(X) = 2N \cdot \ln(N)$$
$$\sigma(X) = 0.65N.$$

, then Let Z be the standardized random variable : $Z = \dfrac{X - E(X)}{\sigma(X)}$

$\Rightarrow$ Z means the number of standard deviations that X is away from the mean. So, the probability that X is more than 10^9 compares is equivalent to the probability that Z is more than : $\dfrac{10^9 - 2N \cdot \ln N}{0.65N}$.

Using $N = 10^6$, $Z > \dfrac{10^9 - 2 \cdot 10^6 \cdot \ln 10^6}{0.65 \cdot 10^6} = \underline{12.53}$.

Now, apply Chebyshev's inequality :

$$P(|Z| > k) < 1/k^2 \text{, where } k = 12.53.$$

$$\therefore P(Z > 12.53) < 1/12.53^2 = \underline{0.0065}$$

Which means the probability that X is more than 10^9 compares is less than 0.0065. So, at least 99.35% confidence that # of compares used by quicksort will not exceed 100 million.

3. (1 points) Show that the solution to $T(n) = 2T\left(\left\lfloor\frac{n}{2}\right\rfloor + 17\right) + n$ is $O(n \lg n)$.

Hint: Show that $T(n) \le c(n-a)\lg(n-a)$ for some constants a and b.

Assume $T(n) \le c \cdot (n-a) \cdot \lg(n-b)$ for some constants a, b and c.

Now we can apply it to the following:

$$T(n) = 2 \cdot T(\lfloor \tfrac{n}{2} \rfloor + 17) + n$$

$$\le 2 \cdot c \cdot (\lfloor \tfrac{n}{2} \rfloor + 17 - a) \cdot \lg(\lfloor \tfrac{n}{2} \rfloor + 17 - b) + n.$$

$$\le 2 \cdot c \cdot (\tfrac{n}{2} + 18 - a) \cdot \lg(\tfrac{n}{2} + 18 - b) + n.$$

$$= c \cdot (n + 36 - 2a) \cdot \lg\left(\frac{n+36-2b}{2}\right) + n.$$

$$= c \cdot (n + 36 - 2a) \cdot \lg(n + 36 - 2b) - c(n + 36 - 2a) \cdot \cancel{\lg 2}^{\,1} + n.$$

$$\le c \cdot (n + 36 - 2a) \cdot \lg(n + 36 - 2b).$$

Which means that we can chose $c = 1$, $a = 18$, $b = 18$

s.t. $T(n) \le n \cdot \lg n$ for $n \ge 68$.

This inequality holds for some initial values of $T(n)$ as well,

so we can say that $\underline{T(n) = O(n \cdot \lg n)}$.

4. Assume there is a max-heap of size 31 whose values are distinct. The largest item in the max-heap must appear at index 1, and the second largest item must be at index 2 or index 3.

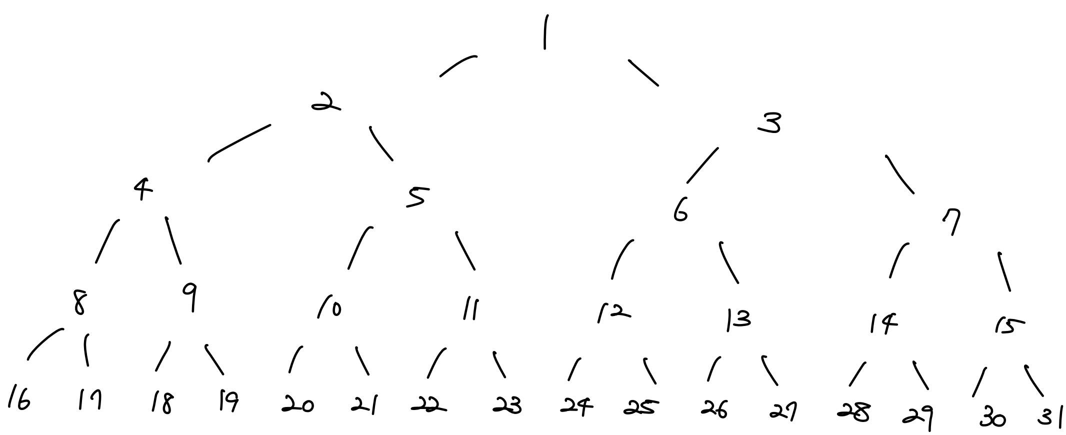
- (1 point) Give the list of indices in the max-heap of size 31 where the k-th largest item (i) can appear, and (ii) cannot appear, for k=2, 3, 4. (assuming the item values to be distinct)) ··· (A)
- (1 point) Give the list of indices in the max-heap of size 31 where the k-th smallest item (i) can appear, and (ii) cannot appear, for k=2, 3, 4. (assuming the item values to be distinct)) ··· (B)

(A) : For k = 2, it can appear at index 2 or 3. So. it cannot appear at index 1 (ancestor) and index in the range [4, 31].

For k = 3, it can apper at index 2 or 3. So it cannot appear at index 1 (ancestor) and index in the range [4, 31].

For k = 4, it can appear at index in the range [4, 7]. So, it cannot appear at index in the range [1, 3] (ancestor) and index in the range [8, 31].

(B) : For k = 2, 3, 4, they can appear at index in the range [16, 31], so it cannot appear at index in the range [1, 15].



5. (2 points) Assume there is an empty max-priority queue A with the following heap procedures.

```
HEAP-EXTRACT-MAX(A)
if A.heap-size < 1
    error "heap underflow"
max = A[1]
A[1] = A[A.heap-size]
A.heap-size = A.heap-size - 1
MAX-HEAPIFY(A, 1)
return max
```

```
MAX-HEAP-INSERT(A, key)
A.heap-size = A.heap-size + 1
A[A.heap-size] = -∞
HEAP-INCREASE-KEY(A, A.heap-size, key)
```

```
HEAP-INCREASE-KEY(A, i, key)
if key < A[i]
    error "new key is smaller than current key"
A[i] = key
while i > 1 and A[PARENT(i)] < A[i]
    exchange A[i] with A[PARENT(i)]
    i = PARENT(i)
```

```
MAX-HEAPIFY(A, i)
l = LEFT(i)
r = RIGHT(i)
if l ≤ A.heap-size and A[l] > A[i]
    largest = l
else largest = i
if r ≤ A.heap-size and A[r] > A[largest]
    largest = r
if largest ≠ i
    exchange A[i] with A[largest]
    MAX-HEAPIFY(A, largest)
```

Illustrate (draw) the max-priority queue of each step of the following operations (draw total 12 priority queues). Assume that items are ordered in a reverse alphabetical manner in our max-priority queue.

- MAX-HEAP-INSERT(A, "P") → MAX-HEAP-INSERT(A, "Q") → MAX-HEAP-INSERT(A, "E") → HEAP-EXTRACT-MAX(A) → MAX-HEAP-INSERT(A, "X") → MAX-HEAP-INSERT(A, "A") → MAX-HEAP-INSERT(A, "M") → HEAP-EXTRACT-MAX(A) → MAX-HEAP-INSERT(A, "P") → MAX-HEAP-INSERT(A, "L") → MAX-HEAP-INSERT(A, "E") → HEAP-EXTRACT-MAX(A)

Step

1)

1
P

2)

1	2
Q	P

3)

1	2	3
Q	P	E

4)

1	2
P	E

5)

X	P	E

6)

X	P	E	A

7)

X	P	M	E	A

8)

P	M	E	A

9)

P	P	M	E	A

10)

P	P	M	L	E	A

11)

P	P	M	L	E	E	A

12)

P	M	L	E	E	A