Exercise Set #3

CMPUT 204, FallL 2019
Department of Computing Science
University of Alberta

Problem 1.
The follwoing exercises from the textbook:
6.1.4, 6.1.5 (page154), 6.4-3 (page 160), and 6.5-1, 6.5-4, and 6.5-9 (pages 164-166).

Problem 2. Consider the following program:

1 4 3 2 5

```
Smallest $(A, f, l)$
**Precondition: $A[f \ldots l]$ is an array of integers, $f \leq l$ and $f, l \in \mathbb{N}$.
**Postcondition: Returns the smallest element of $A[f \ldots l]$.
if $f = l$ then
    return $A[f]$
    2 0
else
    $m = \lfloor \frac{f+l}{2} \rfloor$
    return min(Smallest$(A, f, m)$, Smallest$(A, m+1, l)$)
end if
```

a. Write a recurrence relation which describes the time complexity of this program in terms of the size of array A.

b. Using the master theorem, give a Θ bound on the complexity of this program.

c. Write a non-recursive version of Smallest(A, f, l) (in pseudo-code).

d. Find the worst case running time of this new program and express it in Θ notation.

Problem 3.
Prove asymptotic upper and lower bounds for each of the following recurrences. Assume that in each case, the function is some positive constant for small values of n. You may assume that $n = c^k$ for some constant c that you choose. Make your bounds as tight as you can.

a. $T(n) = 8T(\frac{n}{3}) + n\sqrt{n}$.

b. $T(n) = T(n-1) + \frac{1}{n}$

c. $T(n) = 2T(\frac{n}{2}) + \frac{n}{\lg n}$.

d. $T(n) = T(n-1) + n \lg n$.

Problem 4. Consider the following very simple and elegant(?) sorting algorithm:

```
SomeSort $(A, b, e)$
if $e = b + 1$ then
    if $A[b] > A[e]$ then
        exchange $A[b]$ and $A[e]$
    end if
else if $e > b + 1$ then
    $p \longleftarrow \lfloor \frac{e-b+1}{3} \rfloor$
    SomeSort $(A, b, e - p)$
    SomeSort $(A, b + p, e)$
    SomeSort $(A, b, e - p)$
end if
```

a. Explain why SomeSort correctly sorts its input array A, assuming that $n = e - b + 1$ is the length of the array.

b. Find a recurrence for the worst-case running time of SomeSort. Give a tight (i.e. Θ) asymptotic bound for the worse-case running time of SomeSort (for simplicity assume that $n = 3^k$ for some constant k).

c. By comparing SomeSort with insertion sort, merge sort, heap-sort, and quicksort argue if this simple algorithm is efficient.

Problem 5. A d-ary heap is like a binary heap, except that non-leaf nodes have d children instead of 2 children (with one possible exception for the last leaf).

a. Explain how to implement a d-ary heap in an array and implement the corresponding Parent and Child operation as we had for the binary heap.

```
Parent ($i$)
    return $\lfloor \frac{i-2+d}{d} \rfloor$
Child ($i, j$)
    return $d(i-1)+j+1$
```

b. What is the height of a d-ary heap of n elements in terms of n and d?

c. Write a version of Max-Heapify for a d-ary max-heap and analyze its running time.

d. Write a version of Build-Heap for a d-ary max-heap and analyze its running time.

e. Give an efficient implementation of Extract-Max in a d-ary heap and analyze its running time.

f. Give an efficient implementation of Increase-Key(A, i, k), which first sets $A[i] \longleftarrow \max(A[i], k)$ and then updates the d-ary heap appropriately. Analyze the running time.

g. Write a version of Insert in a d-ary heap and analyze its running time.

Problem 6. Give an algorithm using the techniques and data structures you have seen in the course so far, for the following problem: the input is an array A of size n of integers, find the k largest elements of A and return them. Your algorithm must run in $O(n + k \log n)$. Explain your algorithm and analyze its running time. What is the largest value of k for which your algorithm is still linear in n?

Problem 7. Given a heap A containing n keys and an element x. Propose an efficient algorithm for reporting (say printing) all the keys in A that are greater than or equal to x (note that x is not necessarily in A). The running time of your algorithm should be $O(k)$ where k is the number of elements reported by the algorithm.

Problem 8. An evil king has a cellar containing n bottles of expensive wine and his guards have just caught a spy trying to poison the king's wine. Fortunately, the guards caught the spy after he succeeded in poisoning only one bottle. Unfortunately, they don't know which one. To make matters worse, the poison the spy used was very deadly — just on drop diluted even a billion to one will still kill someone. Even so, the poison works slowly — it takes a full month for the person to die. Design an algorithm that allows the evil king to determine exactly which one of his wine bottles was poisoned in just one month's time while expending at most $O(\log n)$ of his wine tasters. Prove your claims.

1.

6.1-4:

in any of the leaf nodes, that is, the element with index $\lfloor \frac{n}{2} \rfloor + 1$

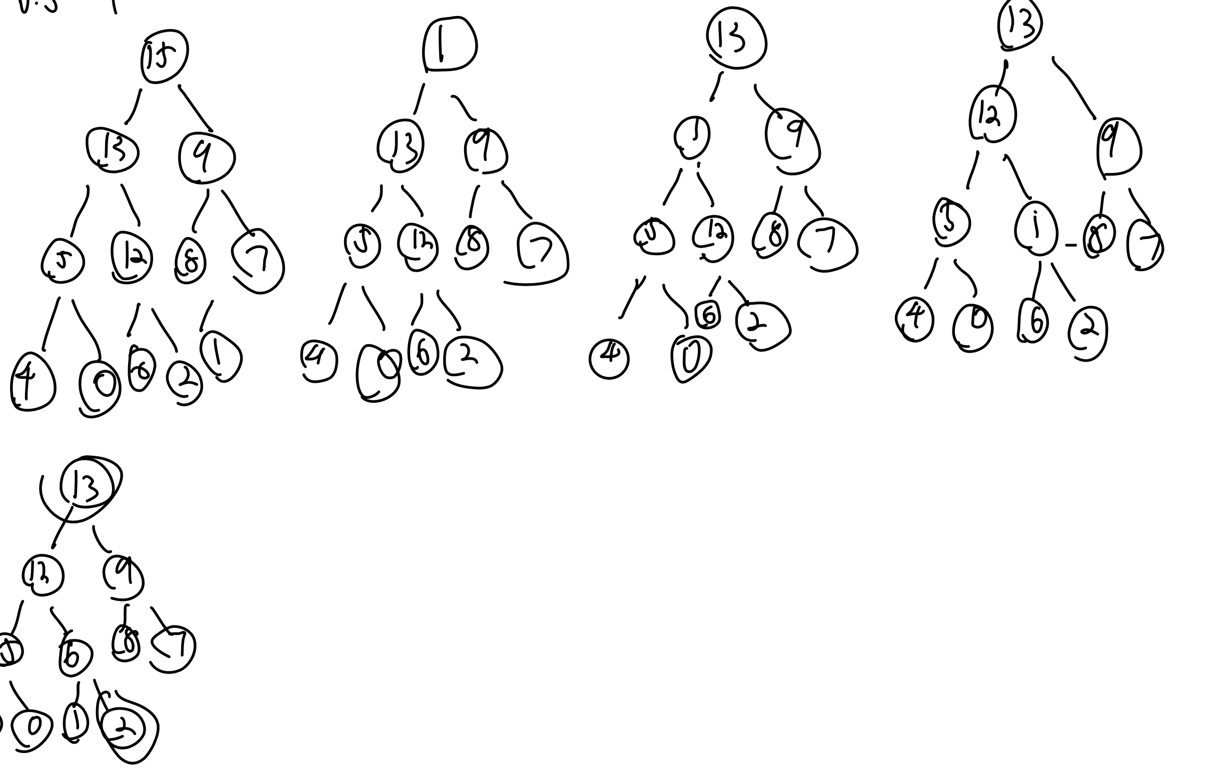
6.1-5

Yes, in a min-heap, the index of a child is always greater than the parent, so the heap property is satisfied.

6.4-3

both are $\Theta(n\lg n)$

6.5-1



6.5-4

to keep the HEAP-INCREASE-KEY still available

6.5-9

Problem 2. Consider the following program: 1 3 4 5

```
Smallest $(A, f, l)$
**Precondition: $A[f \dots l]$ is an array of integers, $f \leq l$ and $f, l \in \mathbb{N}$.
**Postcondition: Returns the smallest element of $A[f \dots l]$.
if $f = l$ then
    return $A[f]$
else
    $m = \lfloor \frac{f+l}{2} \rfloor$
    return min(Smallest$(A, f, m)$, Smallest$(A, m+1, l)$)
end if
```

a. Write a recurrence relation which describes the time complexity of this program in terms of the size of array A.

b. Using the master theorem, give a Θ bound on the complexity of this program.

c. Write a non-recursive version of Smallest(A, f, l) (in pseudo-code).

d. Find the worst case running time of this new program and express it in Θ notation.

a. define $n = l - f + 1$. $T(n)$ is constant for small n and $T(n) = T(\frac{n}{2}) + T(\frac{n}{2}) + C$ where C is a constant

b. $a = 2$ $b = 2$ $d = 0$

since $b^d = 1 < a$

$T(n) = \Theta(n^{\log_b a}) = \Theta(n^{\log_2 2}) = \Theta(n)$

c.
```
smallest ← f
for i ← f+1 to l do
    if A[i] < smallest then
        smallest = i
    end if
end for
return A[smallest]
```

[illegible]

d. $O(n)$ for $n = l - f + 1$. Therefore the running time is $\Theta(n)$

Problem 3.

Prove asymptotic upper and lower bounds for each of the following recurrences. Assume that in each case, the function is some positive constant for small values of n. You may assume that $n = c^k$ for some constant c that you choose. Make your bounds as tight as you can.

a. $T(n) = 8T(\frac{n}{3}) + n\sqrt{n}$.

b. $\underbrace{T(n) = T(n-1) + \frac{1}{n}}$

c. $T(n) = 2T(\frac{n}{2}) + \frac{n}{\lg n}$.

d. $T(n) = T(n-1) + n \lg n$.

a. $a = 8$ $b = 3$ $d = \frac{3}{2}$ $\Theta(n^{\log_b a})$

since $b^d = 3^{\frac{3}{2}} = 5.2 < a$ $\rightarrow \Theta(n^{\log_3 8}) = \Theta(n^2)$

b

$T(n) = T(n-1) + \frac{1}{n}$

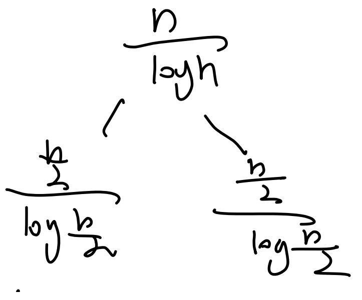
$= T(n-2) + \frac{1}{n-1} + \frac{1}{n}$

$\vdots$

$= T(1) + \frac{1}{2} + \cdots + \frac{1}{n}$

$= \Theta(H_n) = \Theta(\log n)$

c.



$\frac{n}{\log n}$
$\frac{\frac{n}{2}}{\log\frac{n}{2}}$
$\frac{\frac{n}{2}}{\log\frac{n}{2}}$

$T(\frac{n}{2})$

$\frac{n}{\log n - 1}$

$$\frac{\frac{n}{4}}{\log\frac{n}{4}} \quad \frac{\frac{n}{4}}{\log\frac{n}{4}} \quad \frac{\frac{n}{4}}{\log\frac{n}{4}} \quad \frac{\frac{n}{4}}{\log\frac{n}{4}} \quad T\left(\frac{n}{4}\right)$$

$$\frac{n}{\log n - 2}$$

$$\vdots$$

$$\frac{n}{\log n - L}$$

$$T(n) = \sum_{L=0}^{\log_2 n - 1} \frac{n}{\log n - L}$$

$$= n \sum_{L=0}^{\log_2 n - 1} \frac{1}{\log n - L}$$

$$= n \cdot \sum_{i=1}^{\log_2 n} \frac{1}{i} \quad \text{for } i = \log n - L = n \cdot H(\log n) = \Theta(n \log\log n)$$

$$T(n) = n\log n + (n-1)\log(n-1) + \cdots + 1\log 1 + T(0)$$

$$< n\log n + (n-1)\log n + \cdots + 1\log n + T(0)$$

$$= (n + n-1 + \cdots + 1)\log n + T(0)$$

$$= \frac{n(n+1)}{2} \cdot \log n + T(0) \in O(n^2 \log n)$$

Problem 4. Consider the following very simple and elegant(?) sorting algorithm:

```
SomeSort $(A, b, e)$
if $e = b + 1$ then
    if $A[b] > A[e]$ then
        exchange $A[b]$ and $A[e]$
    end if
else if $e > b + 1$ then
    $p \longleftarrow \lfloor \frac{e-b+1}{3} \rfloor$
    SomeSort $(A, b, e - p)$
    SomeSort $(A, b + p, e)$
    SomeSort $(A, b, e - p)$
end if
```

4 6 3 1 2

a. Explain why SomeSort correctly sorts its input array A, assuming that $n = e - b + 1$ is the length of the array.

b. Find a recurrence for the worst-case running time of SomeSort. Give a tight (i.e. Θ) asymptotic bound for the worse-case running time of SomeSort (for simplicity assume that $n = 3^k$ for some constant k).

c. By comparing SomeSort with insertion sort, merge sort, heap-sort, and quicksort argue if this simple algorithm is efficient.

a. Induction: assume SomeSort (A,b,e) correctly sorts the array for any value n.

Case I: when $n \leq 2$, and $b > e$, the two elements are swaped so that they correctly sorted. if $b < e$, the array remains the same and sorted.

case 2 when $n > 2$, suppose $n = 3p$. after the first call, the top p largest elements in b and e-p places between $e-2p$ and $e-p$ in A. Therefore, after the 2nd recursive call, all the top p largest elements of A are in the last p positions. Thus the last call to SomeSort puts the rest of the numbers into the correct locations between b and e-p.

b. line 1-5: constant-time computation, and then recursively call itself for 3 times, each time on an array whose size is $\frac{2}{3}$ of the original array's size.

$T(n) = 3T(\frac{2}{3}n) + \Theta(1)$

master theorem $a=3$ $b=\frac{3}{2}$ $d=0$

$b^d = (\frac{3}{2})^0 = 1 < a$

$\therefore$ $T_n = \Theta(n^{\log_b a}) = \Theta(n^{\log_{\frac{3}{2}} 3})$

C. ~~some~~ sort is the slowest.

Problem 5. A d-ary heap is like a binary heap, except that non-leaf nodes have d children instead of 2 children (with one possible exception for the last leaf).

a. Explain how to implement a d-ary heap in an array and implement the corresponding Parent and Child operation as we had for the binary heap.

```
Parent (i)
    return ⌊(i−2+d)/d⌋        return ⌊(i-2)/d+1⌋
Child (i, j)
    return d(i − 1) + j + 1
```

b. What is the height of a d-ary heap of n elements in terms of n and d?

c. Write a version of Max-Heapify for a d-ary max-heap and analyze its running time.

d. Write a version of Build-Heap for a d-ary max-heap and analyze its running time.

e. Give an efficient implementation of Extract-Max in a d-ary heap and analyze its running time.

f. Give an efficient implementation of Increase-Key(A, i, k), which first sets $A[i] \longleftarrow \max(A[i], k)$ and then updates the d-ary heap appropriately. Analyze the running time.

g. Write a version of Insert in a d-ary heap and analyze its running time.

b. since each node has d children, $\Theta(\log_d n) = \Theta\left(\frac{\lg n}{\lg d}\right)$

c.
```
Max-Heapify (A, i)
    largest ← i
    for j ← 1 to d do
        j ← child (A, i, j)
        if j ≤ heapsize [A] and A[j] > A[largest] then
            largest ← j
    end if
    end for
```

```
if largest ≠ i then
    swap A[i] and A[largest]          O(d log_d n)
    max-Heapify (A, largest)
end if
```

d.
```
Build-Max-Heap (A)
heapsize[A] ← length[A]
for i ← parent(heapsize[A]) down to 1 do
    max-Heapify (A, i)
end for
```

Problem 6. Give an algorithm using the techniques and data structures you have seen in the course so far, for the following problem: the input is an array A of size n of integers, find the k largest elements of A and return them. Your algorithm must run in $O(n + k \log n)$. Explain your algorithm and analyze its running time. What is the largest value of k for which your algorithm is still linear in n?

```
Find Largest (A, n)
    Build-Max-Heap (A);
    for i ← 1 to k
        Answer[i] ← Heap-Extract-Max (A)
end for.
```

Problem 7. Given a heap A containing n keys and an element x. Propose an efficient algorithm for reporting (say printing) all the keys in A that are greater than or equal to x (note that x is not necessarily in A). The running time of your algorithm should be $O(k)$ where k is the number of elements reported by the algorithm.

```
report (A, i, x)
    if A[i] < x then
        return
    print A[i]
    if 2i < heapsize (A) then
        report (A, 2i, x)
```

[illegible]

```
if  2i+1 ≤ heapsize(A) then
    report (A, 2i+1, x)
```

ex.

$T(n) = T(n-1) + n \quad \in O(n^2)$

$T(n) = cn^2 \quad \forall n > n_0$ for some n_0

$T(k+1) = T(k) + k$

$\quad \leq ck^2 + k \quad \leq c(k+1)^2$