cs 5800 - hw2

Yijing Xiao

September 2021

1 Problem 1

Pseudocode 1 MergeSort(non-recursive)

Input: Divide array A into twohalves A-left, A-right. First subarray is A-left[b1..e1], second subarray is A-right[b2..e2]

Output: Sorted array

```
function MergeSort($A, b1, b2, e1, e2$)
    $i \leftarrow 0$
    $temp \leftarrow [\ ]$
    while $b1 \leq e_1$ and $b_2 \leq e_2$ do
        if $A[b_1] \leq A[b_2]$ then
            $temp[i] \leftarrow A[b_1]$
            $i \leftarrow i + 1$
            $b_1 \leftarrow b_1 + 1$
        else
            $temp[i] \leftarrow A[b_2]$
            $i \leftarrow i + 1$
            $b_2 \leftarrow b_2 + 1$
        end if
    end while
    while $b_1 \leq e_1$ do
        $temp[i] \leftarrow A[b_1]$
        $i \leftarrow i + 1$
        $b_1 \leftarrow b_1 + 1$
    end while
    while $b_2 \leq e_2$ do
        $temp[i] \leftarrow A[b_2]$
        $i \leftarrow i + 1$
        $b_2 \leftarrow b_2 + 1$
    end while
    return $temp$
end function
```

2 Problem 2

Since the max-heap float down,the node by consecutively swapping it with higher node below it, the smallest element must be in the leaf node. If we assume that node x contains the smallest element but it is not a leaf, and y is a child node of x, the value of x is greater than to the value of y by the max- heap property. However, this assumption is contradictory. Thus, the smallest element must be in a leaf node.

3 Problem 3

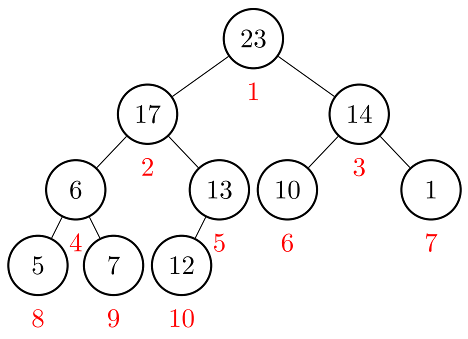

Figure 1: Problem 3

No, it is not. Represent the array as binary tree illustration (shows in figure 1), we can see that the leaf node which contains 7 in position 9 has parent's node which contains 6 in position 4. According to the max-heap property, the value of parent's node should be greater than the value of their children. Thus, this array is not a max-heap.

4 Problem 4

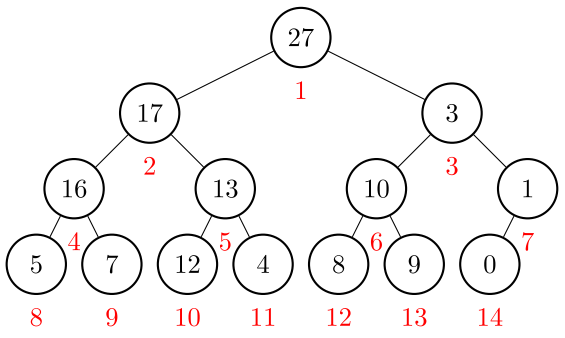

Figure 2: Step 1

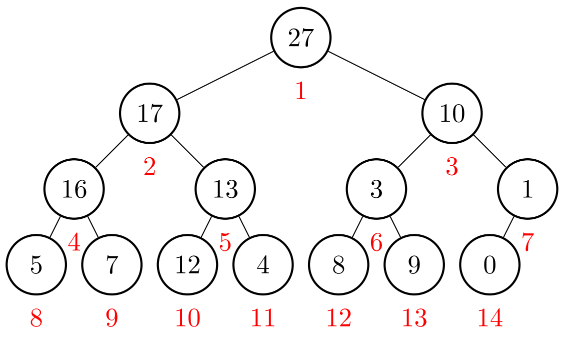

Figure 3: Step 2

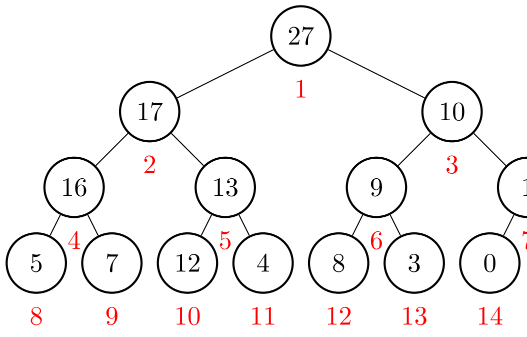

Figure 4: Step 3

5 Problem 5

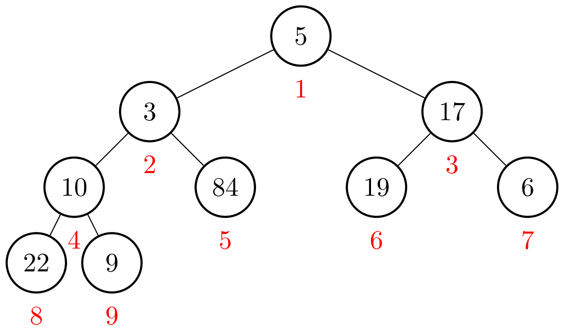

Figure 5: Step 1

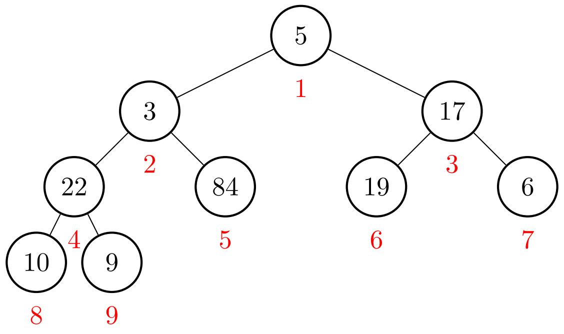

Figure 6: Step 2

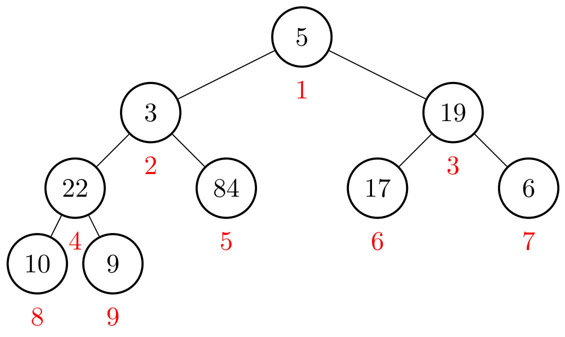

Figure 7: Step 3

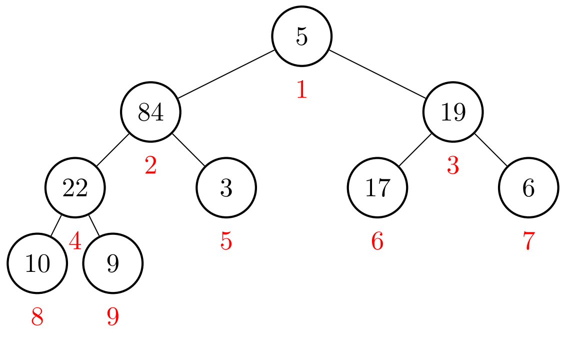

Figure 8: Step 4

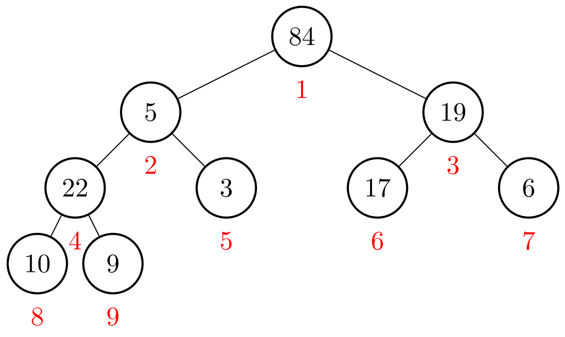

Figure 9: Step 5

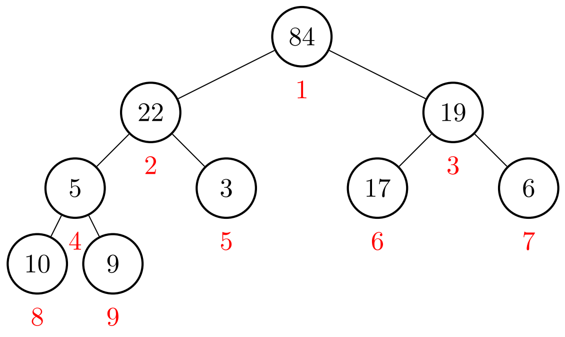

Figure 10: Step 6

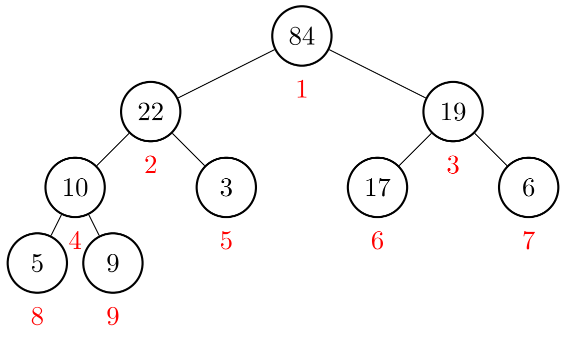

Figure 11: Step 7

6 Problem 6

6.1 a. How would you represent a d-ary heap in an array?

Assuming 0 based indexing of array, an array represent a d-ary heap such that for any node we consider:
Parent of the node at index i (except root node) is located at index $\frac{i-1}{d}$
Children of the node at index i are at indices $(d*i)+1, (d*i)+2, ..., (d*i)+d$

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

$\log_d n$

6.3 c. Give an efficient implementation of Extract-Max in a d-ary max-heap. Analyze its running time in terms of d and n.

Pseudocode 2 Extract-Max

```
function EXTRACT-MAX($A$)
    $max \leftarrow A[0]$
    $A[0] \leftarrow A[n-1]$
    $n \leftarrow n-1$
    D-ARY MAX-HEAP(A, n, d)
end function
```

A d-ary max-heap stores the largest element in its root. The Extract-Max returns the root node, so we copies the last node to the first and calls d-ary Max-heap to maintain the heap property. The number of times this recursively calls itself is bounded by the height if the d-ary heap. Therefore the running time is $O(d * log_d n)$

6.4 d. Give an efficient implementation of Insert in a d-ary max-heap. Analyze its running time in terms of d and n.

Pseudocode 3 Insertion

```
function INSERT($A, n, element$)
    $A[n] \leftarrow element$
    $n \leftarrow n + 1$
    while $n > 1$ and $A[parent(n)] < A[n]$ do
        exchange $A[n]$ with $A[parent(n)]$
        $n \leftarrow Parent(n)$
    end while
end function
```

The running time is $O(log_d n)$ because the while loop runs at most as many times as the height of the d-ary array.

6.5 e. Give an efficient implementation of Increase-Key, which flags an error if $k < A[i]$, but otherwise sets $A[i] = k$ and then updates the d-ary max-heap structure appropriately. Analyze its running time in terms of d and n.

Pseudocode 4 Increase-Key

```
function INCREASE-KEY($A, i, k$)
    if $k < A[i]$ then
        error"k is smaller than $A[i]$"
    end if
    $A[i] \leftarrow k$
    while $i > 1$ and $A[parent(i)] < A[i]$ do
        exchange $A[i]$ with $A[parent(i)]$
        $i \leftarrow Parent(i)$
    end while
end function
```

The running time is $O(log_d n)$ because the while loop runs at most as many times as the height of the d-ary array.

7 Problem 7

According to the definition of Θ notation, there exists some positive constants c_1, c_2 so that the $\Theta(n)$ term is between $c_1 n$ and $c_2 n$.
Solution: we guess that $T(n) \leq O(n^2)$

$$\begin{aligned}
&T(n) \leq c_1(n-1)^2 + \Theta(n)\\
&\leq c_1(n-1)^2 + c_0 n\\
&\leq c_1(n^2 - 2n + 1) + c_0 n\\
&\leq c_1 n^2 - 2c_1 n + c_1 + c_0 n\\
&\leq c_1 n^2 - (2c_1 - c_0)n + c_1\\
&\leq c_1 n^2 \text{ for } n \geq 1 \text{ and } 2c_1 \geq c_0
\end{aligned}$$

$$\begin{aligned}
&T(n) \geq c_2(n-1)^2 + \Theta(n)\\
&\geq c_2(n-1)^2 + c_3 n\\
&\geq c_2(n^2 - 2n + 1) + c_3 n\\
&\geq c_2 n^2 - 2c_2 n + c_2 + c_3 n\\
&\geq c_2 n^2 - (2c_2 - c_3)n + c_2\\
&\geq c_2 n^2 \text{ for } n \geq 1 \text{ and } 2c_2 \geq c_3
\end{aligned}$$

Therefore, $T(n) = \Theta(n^2)$

8 Problem 8

Pseudocode 5 QuickSort

```
function QuickSort($A, p, r$)
    if $p < r$ then
        $q$ = Partition($A, p, r$)
        QuickSort($A, p, q - 1$)
        QuickSort($A, q + 1, r$)
    end if
end function

function Partition($A, p, r$)
    $x \leftarrow A[r]$
    $i \leftarrow p - 1$
    for $j = p$ to $r - 1$ do
        if $A[j] \leq x$ then
            $i \leftarrow i + 1$
            exchange $A[i]$ with $A[j]$
        end if
    end for
    exchange $A[i + 1]$ with $A[r]$
    return $i + 1$
end function
```

If all elements in the array have the same value, $Partition(A, q, r)$ returns r which means the partition will always occur at the last position of the array. Since the execution time of the partition process only depends on n, $T_{partition}(n) = \Theta(n)$

$T(n) = T(n - 1) + T(0) + T_{partition}(n)$
$= T(n - 1) + \Theta(n)$
$= T(n - 2) + 2\Theta(n)$
$= T(n - 3) + 3\Theta(n)$
...
$= T(n - n) + n\Theta(n)$
$= \Theta(n^2)$

Therefore, the running time is $\Theta(n^2)$

9 Problem 9

According to the pseudo-code of QuickSort in last question, we could know that if the array is sorted in decreasing order, then the pivot element is leaa than all the other elements. We've already know $T_{partition} = \Theta(n)$ and $T(n) = T(n-1)+T(0)+T_{partition}(n) = \Theta(n^2)$ in last question. Therefore, the running time still is $\Theta(n^2)$

10 Problem 10

We guess that $T(n) \leq cn^2$ for some positive constant c, substituting this guess into recurrence, we obtain:

$$T(n) \leq \max_{0 \leq q \leq n-1} (cq^2 + c(n - q - 1)^2) + \Theta(n)$$

$$= c * \max_{0 \leq q \leq n-1} (q^2 + (n - q - 1)^2) + \Theta(n)$$

The expression $q^2+(n-q-1)^2$ achieves a maximum over the parameter's range $0 \leq q \leq n-1$ at either endpoint. To verify this claim, note that the second derivative of the expression with respect to q is positive. This observation give us the bound $\max_{0\leq q\leq n-1}(cq^2+c(n-q-1)^2) \leq (n-1)^2 = n^2-2n+1$. Continuing with our bounding of $T(n)$, we obtain:

$$T(n) \leq cn^2 - c(2n-1) + \Theta(n)$$
$$\leq cn^2$$

Since we can pick the constant c large enough so that that $c(2n-1)$ term dominates the $\Theta(n)$ term. Thus, $T(n) = O(n^2)$. Since $T(n) = \Omega(n^2)$, the final answer is $T(n) = \Theta(n^2)$

11 Problem 11

In the most even possible split, *Partition* function produces two sub-problems, each of size no more than $\frac{n}{2}$, since one is size $(\frac{n}{2})$ and one of size $(\frac{n}{2})-1$. In this case, QuickSort running time is:

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

According to the simple Master Theorem, $a=2, b=2, c=1, \log_b a = log_2 2 = 1. c = \log_b a$ so case 2. This recurrence has the solution $T(n) = \Theta(n \lg n)$ which also is bounded by $\Omega(n \lg n)$

12 Problem 12

$$f(q) = q^2 + (n-q-1)^2$$
$$f'(q) = 2q + 2(n-q-1)*(-1) = 4q - 2n + 2$$
$$f''(q) = 4 > 0$$

The second derivative is greater than 0 which means that $f'(q)$ is negative left of $f'(q) = 0$ and positive right of it, which means that this is a local minimal. In

$$f'(q) = 0 \Rightarrow 4q - 2n + 2 = 0 \Rightarrow q = \frac{1}{2}n - \frac{1}{2}$$

In this case, $f(q)$ is decreasing in the beginnings of the interval and increasing in the end, which means that the maximal value must only be the endpoints. The endpoints are $q = 0$ and $q = n-1$ which can make the expression achieves maximum.