Fundamental Algorithms, Home work - 3

Tulsi Jain

1. First Problem

1.1

Array, A =

19	2	11	14	7	17	4	3	5	15

After initial heapification, A =

19	15	17	14	7	11	4	3	5	2

Swap first element with last, A =

2	15	17	14	7	11	4	3	5	19

After heapification of A.length -1 elements, A =

17	15	11	14	7	2	4	3	5	19

Swap first element with second last, A =

5	15	11	14	7	2	4	3	17	19

After heapification of A.length - 2 elements, A =

15	14	11	5	7	2	4	3	17	19

Swap first element with third last, A =

3	14	11	5	7	2	4	15	17	19

After heapification of A.length - 3 elements, A =

14	7	11	5	3	2	4	15	17	19

Swap first element with fourth-last, A =

4	7	11	5	3	2	14	15	17	19

After heapification of A.length - 4 elements, A =

11	7	4	5	3	2	14	15	17	19

Swap first element with fivth-last, A =

2	7	4	5	3	11	14	15	17	19

After heapification of A.length - 5 elements, A =

7	5	4	2	3	11	14	15	17	19

Swap first element with six-last, A =

3	5	4	2	7	11	14	15	17	19

After heapification of A.length - 6 elements, A =

5	3	4	2	7	11	14	15	17	19

Swap first element with seventh-last, A =

2	3	4	5	7	11	14	15	17	19

After heapification of A.length - 7 elements, A =

4	3	2	5	7	11	14	15	17	19

Swap first element with eigth-last, A =

2	3	4	5	7	11	14	15	17	19

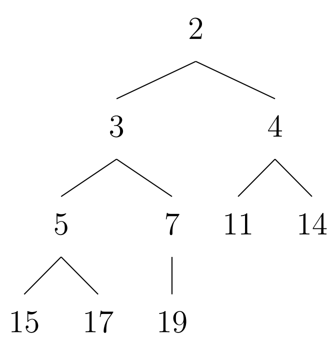
After heapification of A.length - 8 elements, A =

3	2	4	5	7	11	14	15	17	19

Swap first element with ninth-last, A =

2	3	4	5	7	11	14	15	17	19

Only one element is left hence. no more max heapification is required. Array is sorted.



We got the, sorted array at the end.

2. Second Problem

Array, A = [4, 6, 3, 5, 0, 5, 1, 3, 5, 5]

As elements vary from 0 to 6 in A, we would initialize a array, C of size 7 with zero as starting value.
Array, C = [0,0,0,0,0,0,0]
Traverse Array A, to fill C as C[i] denote the number of occurrence of i-1 in A.
Array, C = [1,1,0,2,1,4,1]
Convert C to accumulative sum the array, C = [1,2,2,4,5,9,10]
Now the final part

New Array, B of size A.length

For loop from A.length to 1;
int x = A[i];
int y = C[x + 1];
B[y] = x;
C[x+1] = y -1;

Starting B = [0,0,0,0,0,0,0,0,0,0], C = [1,2,2,4,5,9,10]
After one iteration, B = [0, 0, 0, 0, 0, 0, 0, 0, 5, 0], C = [1, 2, 2, 4, 5, 8, 10]
After second iteration, B = [0, 0, 0, 0, 0, 0, 0, 5, 5, 0], C = [1, 2, 2, 4, 5, 7, 10]
After third iteration, B = [0, 0, 0, 3, 0, 0, 0, 5, 5, 0], C = [1, 2, 2, 3, 5, 7, 10]
After fourth iteration,B = [0, 1, 0, 3, 0, 0, 0, 5, 5, 0], C = [1, 1, 2, 3, 5, 7, 10]
After fifth iteration, B = [0, 1, 0, 3, 0, 0, 5, 5, 5, 0], C = [1, 1, 2, 3, 5, 6, 10]
After six iteration, B = [0, 1, 0, 3, 0, 0, 5, 5, 5, 0], C = [0, 1, 2, 3, 5, 6, 10]
After seventh iteration, B = [0, 1, 0, 3, 0, 5, 5, 5, 5, 0], C = [0, 1, 2, 3, 5, 5, 10]
After eighth iteration, B = [0, 1, 3, 3, 0, 5, 5, 5, 5, 0], C = [0, 1, 2, 2, 5, 5, 10]
After ninth iteration, B = [0, 1, 3, 3, 0, 5, 5, 5, 5, 6], C = [0, 1, 2, 2, 5, 5, 9]
B = [0, 1, 3, 3, 4, 5, 5, 5, 5, 6], C = [0, 1, 2, 2, 4, 5, 9]

We got the B as sorted array in the last.

3. Third Problem

Array, A = [392, 517, 364, 931, 726, 912, 299, 250, 600, 185]

As per this radix algorithm least important digit should be sorted first while maintaining the order in case of equal value. In given question maximum number of digit are three hence we would sort three times starting from right most digit. While doing the intermediate sorting it is mandatory to use the a stable algorithms.

Following are the iteration process.

Input
392
517
364
931
726
912
299
250
600
185

Sort by Last digit
250
600
931
392
912
364
185
726
517
299

Sort by middle digit
600
912
517
726
931
250
364
185
392
299

Sort by Left most digit
185
250
299
364
392
517
600
726
912
931

We got the sorted array after sorting the left most digit.

4. Fourth Problem

Array, A = [0.88, 0.23, 0.25, 0.74, 0.18, 0.02, 0.69, 0.56, 0.57, 0.49]

As per this bucket algorithm, array element are put into buckets then each bucket is sorted using sorting algo like insertion sort

Let's create 10 buckets numbered from 0 to 9. Add ith element into [10*array[i]] numbers buckets.

0 → 0.02
1 → 0.18
2 → 0.23, 0.25
3 →
4 → 0.49
5 → 0.56, 0.57
6 → 0.69
7 → 0.74
8 → 0.88
9 →

After sorting each bucket individually by insertion sort.

0 → 0.02
1 → 0.18
2 → 0.23, 0.25
3 →
4 → 0.49
5 → 0.56, 0.57
6 → 0.69
7 → 0.74
8 → 0.88
9 →

After merging all buckets from top to bottom.

Array, B = [0.02, 0.18, 0.23, 0.25 0.49, 0.56, 0.57,, 0.69, 0.74, 0.88]

5. Fivth Problem

Represent d-ary heap in an array. Let's say an array A. For Root Node, A[1] and its children are A[2] to A[d+1]
For Node, A[2], its children are A[d+2] to A[2d+1]
For Node, A[3], its children are A[2d+2] to A[3d+1]
For ith Node, its children are A[(i-1)d+2] to A[id+1]
Similarly
Parent of A[2] to A[d+1] is A[1]
Parent of A[d + 2] to A[2d+1] is A[2]
Parent of A[2d + 2] to A[3d+1] is A[3]
Parent of A[di - d + 2] to A[id+1] is A[i]

5.1 Parent of j-th node in a d-ary heap

For $(i-1)d+2 <= j <= id+1$, Parent of jth would be ith
Hence, Parent of jth = $\left\lfloor \frac{(j+d-2)}{d} \right\rfloor$th. Let's check if this hold
When $j = (i-1)d+2$, then = $\lfloor \frac{id-d+2+d-2}{d} \rfloor = i$
When $j = id+1 = \lfloor \frac{id+1+d-2}{d} \rfloor = i$
It holds for both end value hence it would also hold for all intermediate value

Parent of jth node = $\lfloor \frac{j+d-2}{d} \rfloor$th element

5.2 j-th child of i-th node in a d-ary heap.

As showed, earlier first children of ith node is [(i-1)d + 2]th element. Hence jth children of ith node is [(i-1)d + 2 + j - 1]th element.
Answer is: [(i-1)d + j + 1]th element

5.3 Number of nodes of height h in an n-element d-ary heap

Let's first compute number of leaves in d-array heap with number of elements n. it would be as follows. (Let's say heap has a height of, h);

$$\#leaves = n - \frac{d^h - 1}{d-1} + \frac{\frac{d^{h+1}-1}{d-1} - n}{d}$$

By Above equation # leaves is $\lceil \frac{dn-n}{d} \rceil$
Above is proved by rough calculation

Induction proof for number of nodes of height, h
Hypothesis Step: Now guess number of nodes of height h is $\lceil \frac{dn-n}{d^{h+1}} \rceil$
Let's prove this by induction,
Base Case: When h =0; of nodes is $\lceil \frac{dn-n}{d^{0+1}} \rceil = \lceil \frac{dn-n}{d^{1}} \rceil$, hence, it holds
Inductive step: Prove this is for height, h. Let's N_h denotes the numbers of nodes of height, h.
$N_h = \left\lceil \frac{N_{h-1}}{d} \right\rceil$
Using Induction hypothesis
$N_{h-1} = \lceil \frac{dn-n}{d^h} \rceil$
$N_h = \left\lceil \frac{\lceil \frac{dn-n}{d^h} \rceil}{d} \right\rceil$
$N_h = \lceil \frac{dn-n}{d^{h+1}} \rceil$
Hence our induction proof holds.
Number of nodes of height h is, $N_h = \lceil \frac{dn-n}{d^{h+1}} \rceil$

5.4 Height of an n-element d-ary heap

Let's say it has height, h. As heap is almost complete tree. We can say

$$\frac{d^h-1}{d-1} + 1 <= n < \frac{d^{h+1}-1}{d-1} + 1$$

$$\frac{d^h-1}{d-1} <= n-1 < \frac{d^{h+1}-1}{d-1}$$

$$d^h - 1 <= (n-1)(d-1) < d^{h+1} - 1$$

$$d^h <= (n-1)(d-1) + 1 < d^{h+1}$$

$$d^h <= nd - n - d + 2 < d^{h+1}$$

$$h <= log_d(nd - n - d + 2) < h + 1$$

h is always integer, $\log_d(nd - n - d + 2)$ will always be greater than equal to h and less than h+1 Hence h = $\lfloor \log_d nd - n - d + 2 \rfloor$

6. Six Problem

6.1 Unordered array

6.1.1 Insert k

Algorithm 1 Insert Element

```
function INSERT(A, k, n)                ▷ where Array A, insert k, size n
    A[n+1] = k;
end function
```

6.1.2 GetMin & return

Algorithm 2 Find Minimum

```
function FINDMINIMUM(A)                 ▷ where Array A, size n
    int min = A[1];
    for i = 2 to n do
        if min > A[i] then
            min = A[i]
        end if
    end for
    return min
end function
```

6.1.3 ExtractMin, remove & return

Algorithm 3 Find Minimum and remove

```
function RETURNMIN(A, n)                ▷ where Array A, size n
    min = A[1]; indexMin = 1;
    for i = 2 to n do
        if min > A[i] then
            min = A[i];
            indexMin = i
        end if
    end for
    for i = indexMin to n − 1 do
        A[i] = A[i+1];
    end for
    return min
end function
```

6.2 Ordered array, Assume array is ascending

6.2.1 Insert k

Algorithm 4 Insert Element

```
function INSERT(A, k, n)                    ▷ where Array A, insert k, size n
    index = 1;
    for i = 1 to n do
        if k > A[i] then
            index++;
        else
            break;
        end if
    end for
    temp = k
    for i = index to n do
        temp2 = A[i]
        A[i] = temp
        temp = temp2
    end for
end function
```

6.2.2 GetMin & return

Algorithm 5 Find Minimum

```
function FINDMINIMUM(A, n)                  ▷ where Array A, size n
    return A[1]
end function
```

6.2.3 ExtractMin, remove & return

Algorithm 6 Find Minimum

```
function EXTRACTMIN(A, n)                   ▷ where Array A, size n
    int min = A[1]
    for i = 2 to A.length do
        A[i-1] = A[i]
    end for
    return min
end function
```

6.3 Unordered linked list

6.3.1 Insert k

Algorithm 7 Insert Element

```
function INSERT(node, k)                    ▷ where Linkedlist node, int k
    newNode = new Node(k);
    newNode.next = node;
    startingNode = newNode;
end function
```

6.3.2 GetMin & return

Algorithm 8 Find Minimum

```
function FINDMINIMUM(node)                  ▷ where Linked-list node
    temp = node.value;
    startingNode = node
    while node != null do
        if temp < node.value then
            temp = node.value;
        end if
        node = node.next;
    end while
    return temp;
end function
```

6.3.3 ExtractMin, remove & return

Algorithm 9 Find Minimum

```
function ExtractMin(node)                          ▷ where Linked-list node
    startingNode = node
    tempnode = node
    temp = node.value
    while node.next != null do
        if temp < node.next.value then
            temp = node.next.value;
            tempnode = node;
        end if
        node = node.next;
    end while
    tempnode.next = tempnode.next.next
    return temp;
end function
```

6.4 Ordered linked list, Assume array is ascending

6.4.1 Insert k

Algorithm 10 Insert Element

```
function Insert(node, k)                  ▷ where Linked-list node, insert k
    startingNode = node
    if k < node.value then
        tempNew = new Node(k);
        tempNew.next = node.next;
        return
    end if
    while node.next != null do
        if k < node.next.value then
            break;
        end if
        node = node.next;
    end while
    tempNode = node.next;
    tempNew = new Node(k);
    node.next = tempNew;
    tempNew.next = tempNode;
end function
```

6.4.2 GetMin & return

6.4.3 ExtractMin, remove & return

Algorithm 11 Find Minimum

```
function FindMinimum(node)                    ▷ where Linked-list node
    return node.value;
end function
```

Algorithm 12 Find Minimum

```
function ExtractMin(node)                     ▷ where Linked-list node
    min = node.value;
    startingNode = node.next;
    return min;
end function
```

6.5 Min-heap

6.5.1 Insert k

Algorithm 13 Insert Element

```
function Insert(A, n, k)          ▷ where A Array, n size, k insertelement
    A[n + 1] = k
    MakeHeap(A, n+1, n+1)
end function
function MakeHeap(A, i, n)        ▷ where Array, tobeMiniHeapify i, size n
    parent = 2*i
    smallest = indexOfMin (A[parent], A[i])
    if indexOfMin =! parent then
        temp = A[parent];
        A[parent] = A[smallest];
        A[smallest] = temp;
        MakeHeap(A, parent, n)
    end if
end function
```

6.5.2 GetMin & return

6.5.3 ExtractMin, remove & return

Algorithm 14 Find Minimum

```
function FindMinimum(A, n)                          ▷ where A Array, n size
    return A[1];
end function
```

Algorithm 15 Find Minimum

```
function FindMinimum(A, n)                          ▷ where A Array, n size
    min = A[1];
    A[1] = A[n];
    MinHeapify(A, 1, n-1);
    return min;
end function
function MinHeapify(A, i, n)   ▷ where A Array, i tobeMinheapify Node, n end
    left = 2*i;
    right = 2*i +1;
    smallest = indexOfMin (A[i], A[left], A[right])
    if smallest ! = i then
        temp = A[i]
        A[i] = A[smallest]
        A[smallest] = temp
        MinHeapify(A, smallest, n)
    end if
end function
```

Time complexity

Implemented data structures type	insert k	Find minimun	Find minimun and remove
Unordered array	O(1)	O(n)	O(n)
Ordered array	O(n)	O(1)	O(n)
Unordered linked list	O(1)	O(n)	O(n)
Ordered linked list	O(n)	O(1)	O(1)
Min-heap	logn	O(1)	logn