Design and Analysis of Algorithms

Assignment-03

Dr. Ram P Rustagi
Sem IV (2020-Even)
Dept of CSE, KSIT
rprustagi@ksit.edu.in

Assignment Logistics

- Same group as that of HA01. To change, make a request.
- All assignments are to be submitted online in github.
  – Program (java/C/C++/python) should run on Linux.
- Program should adhere to following
  - Use command line arguments for parameter passing
    - Avoid any hard coding of parameter values.
  - Program should not crash under any circumstances
  - It should have proper indentation for readability
  - Use proper variable names to indicate meaning
    - Avoid use of cryptic variable names e.g. `a,b,c,x,y`

Assignment Logistics

- There are total of 12 programming questions.
  - All Qs requires use of Divide/Conquer or Decrease/ Conquer apporach. (No brute force)
- Each group/team is assigned one of the questions and need to submit the same question. The assignment question number is same as that for HA01.
- A team is encouraged to do other non-assigned questions to help improve learning.
  - Team may be given bonus marks (as per the discretion of the instructor). The bonus marks, if given, may count towards previous/future assignments.
- Plagiarism (copy) will result in 0 marks for all
- Any partial code from net (googling) should be cited with URL along with explanation

Assignment Submission Details

- Each submission (on github) should include
  - The program name should preferably be as e.g.
    - Q01: `Asn03G01P01.java/c/cpp/py ...`
    - :
    - Q12: `Asn03GxxP12.java/c/cpp/py ...`
  - `Readme.txt`: should contain
    1. Team details (Names, USN)
    2. Contribution of each team member
    3. Instructions to run the program
    4. Challenges faced and how did you address these
    5. What did you learn from this assignment
  - `Output.txt`
    1. Output of program with your sample data
    2. Total number of basic (key) operations i.e. computation of time complexity.

Assignment Input

- If any assignment question involves graph, the graph data is specified in an input file with following syntax.

```
<N> <M>
u v c
:
:  (M such lines)
```

- First line contains two values `N` (number of vertices) and `M` (number of edges). After first line, there are `M` lines, each one having `3` values `a b c`, implying there is a directed edge from `a` to `b` with cost `c`. An example of graph is below. For undirected graph,each edge appears twice.

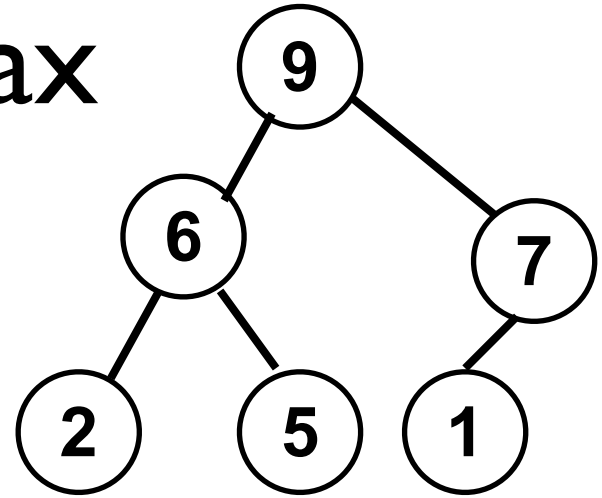
```
3 4
1 2 3
1 3 6
2 4 5
3 2 1
```

Q01: Implement HeapSort (Ascending)

- Given an input sequence of numbers build a Maxheap and then perform heapsort operation inplace (i.e. no new array is to be used)(
- For example, the sequence 2, 10, 3, 1, 4, 8, 5, 6, 11, 9
- Hint: Perform Heapify operation
  - Put all the elements in an array, and starting from `⌊n/2⌋` to `1`, perform heapify i.e. build a maxheap.
  - Now perform deletemax() operation till heap becomes empty. Deletemax() returns the largest element and put the element at the end of heap.
    - Thus, largest element (returned by first deletemax) will be stored at `A[n]`
    - 2nd largest element will be stored at `A[n-1]`
- i/p: comma separated values e.g. 2,10,3,1,4,8,5,6,11,9

Series of DeleteMax

-	2	9	1	6	5	7
-	9	6	7	2	5	1

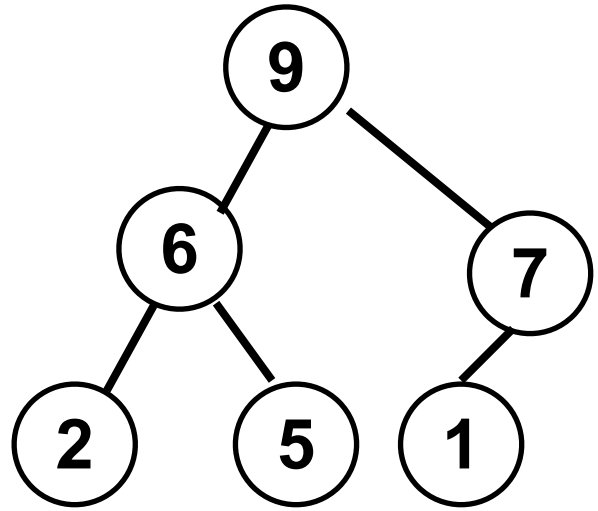


- Deletemax i.e. delete 9

-	7	6	1	2	5	9

- Deletemax i.e. delete 7

-	6	5	1	2	7	9



- Deletemax i.e. delete 6

-	5	2	1	6	7	9

- Deletemax i.e. delete 5

-	2	1	5	6	7	9

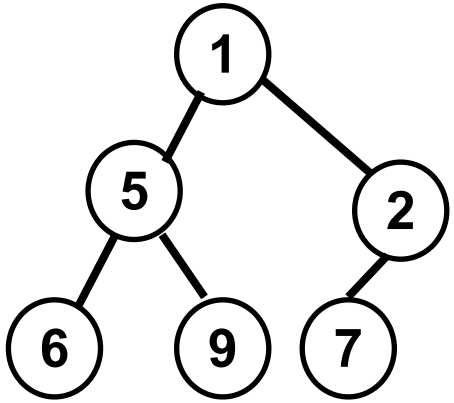
Q02: Implement HeapSort (Decending)

- Given an input sequence of numbers build a Minheap and then perform heapsort operation inplace (i.e. no new array is to be used)(
- For example, the sequence 2, 10, 3, 1, 4, 8, 5, 6, 11, 9
- Hint: Perform Heapify operation
  - Put all the elements in an array, and starting from $\lfloor n/2 \rfloor$ to 1, perform heapify i.e. build a maxheap.
  - Now perform deletemin() operation till heap becomes empty. Deletemin() returns the smallest element and put the element at the end of heap.
    - Thus, smallest element (returned by first deletemin) will be stored at `A[n]`
    - 2nd smallest element will be stored at `A[n-1]`
- i/p: comma separated values e.g. 2,10,3,1,4,8,5,6,11,9

Series of DeleteMin

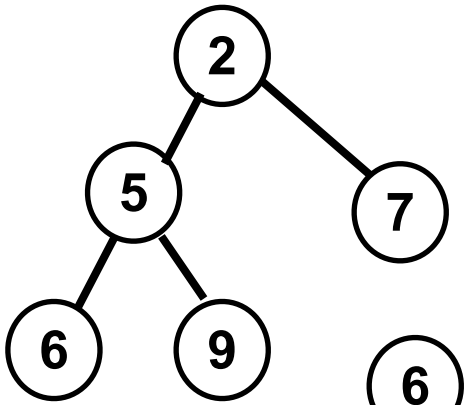
-	2	9	1	6	5	7

-	1	5	2	6	9	7



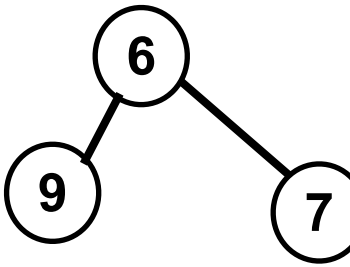
- Deletemin i.e. delete 1

-	2	5	7	6	9	1



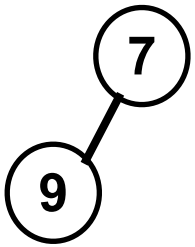
- Deletemin i.e. delete 2

-	5	6	7	9	2	1



- Deletemin i.e. delete 5

-	6	9	7	5	2	1



- Deletemin i.e. delete 6

-	7	9	6	5	2	1

Q03: Heap Insertion

- Take each letter of your full name in an array and then build a maxheap of letters of your name.
  - Once this heap is build, take letters of name of your teammate, and insert these letters into this heap.
- Input: <name1> <name2> <name3>
- Example: consider the first name as "`computer`" and

-	c	o	m	p	u	t	e	r

- After heapification (Maxheap), the array becomes

-	u	r	t	p	o	m	e	c

- After inserting "`science`" the maxheap array becomes

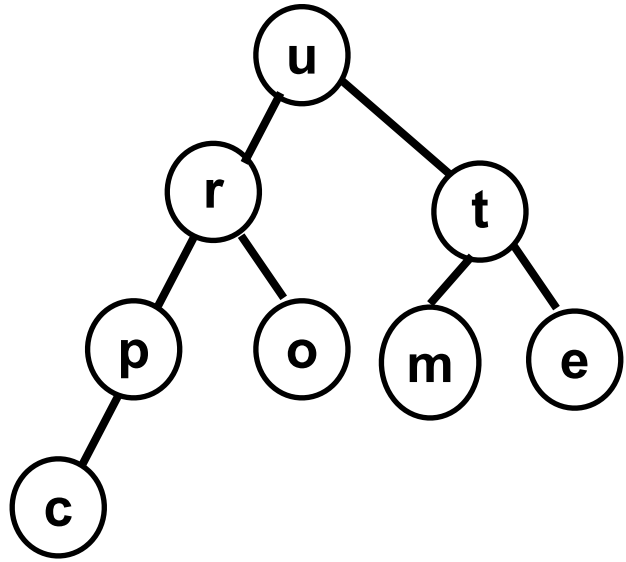
-	u	s	t	r	o	n	e	c	p	c	i	e	m	c	e

Series of Insert

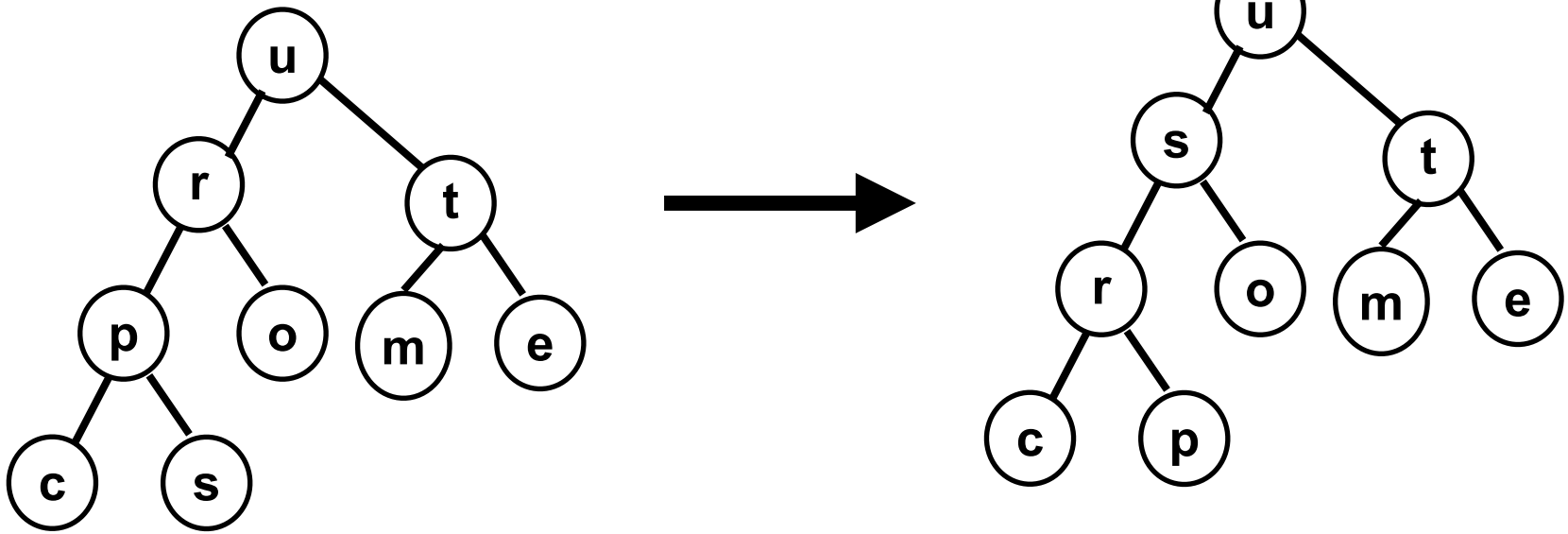
-	c	o	m	p	u	t	e	r

- MaxHeap

-	u	r	t	p	o	m	e	c



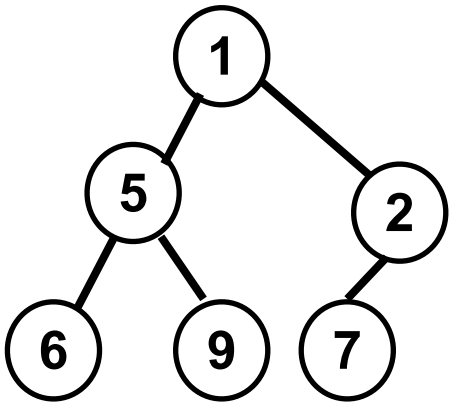
- MaxHeap after insert 's' at the end
  - Perform bubbleup



Q04: HeapSort (Modification)

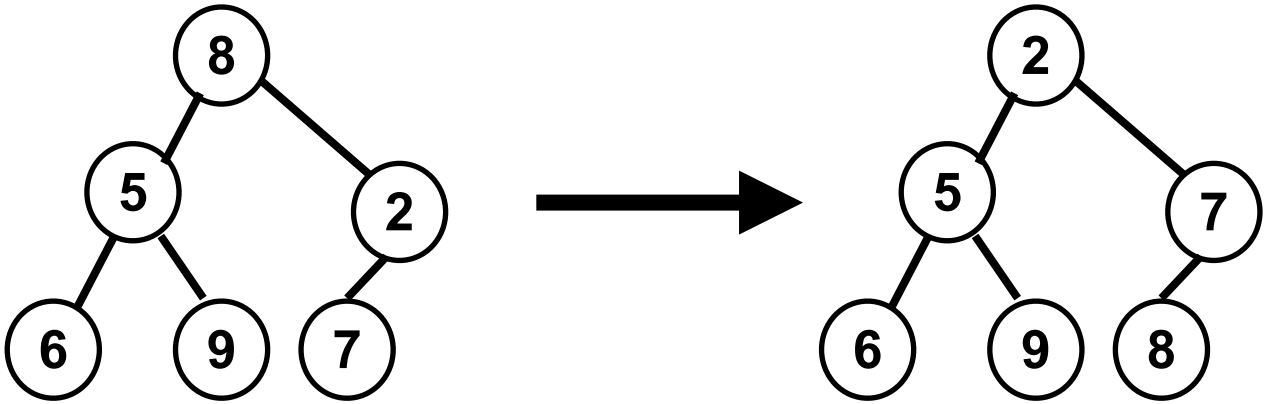
- Given an input sequence of numbers (comma separated) build a Minheap.
  - For example, the sequence 2,10,3,1,4,8,5,6,11,9
- On the current heap, perform the following modification operation
  - The value of the node is increased by 1 if its initial location (index) is even and decreased by 1 if its initial location (index) is odd. The location/index starts from 1. E.g. the new values for the above input will become
    - 1,11,2,2,3,9,4,7,10,10
- i/p: comma separated values e.g. 2,10,3,1,4,8,5,6,11,9

Modification



-	1	5	2	6	9	7

- Change 1 to 8 i.e. increased, thus
  - it has to sinkdown



Q05: Optimal Encoding

- Consider the problem of optimal encoding of names of your team. Consider the letters that occcur in your name including SPACE, and build a Huffman Tree for these, and given an encoding of the names, output the encoding for each letter, name in encoded form.
  - For example, for names:
    - "`computer science and engineering`", the letter frequencies are

```
e: 6
c: 3
i: 3
n: 3
` `:3(Space)
r: 2
g: 2
s: 1
a: 1
d: 1
o: 1
m: 1
p: 1
u: 1
t: 1
```

Optimal Encoding

Q06: Game of 3 Dices

- Consider that a person throws 3 unbiased dices and notes the sum of numbers that appears on dices. You need to guess the sum of dices by asking minimum number of questions. Answer to questions are either **Yes** or **No**. e.g. Question can be: Is the number 4, or 5 or 9? Build a strategy so as to guess the sum in (expected) minimum number of questions.
- Hint: Sum of numbers ranges from 3 to 18 with their respective frequencies as
  3-1,4-3,5-6,6-10,7-15,8-21,9-25,10-27,11-27,12-25,13-21,14-15,15-10,16-6,17-3,18-1
- Build a Huffman tree to answer the right question.
- Note: There are total 16 numbers, and thus using binary search, answer can be obtained using 4 questions. Your answer should be better than 4.

Game of 3 Dices

Q07: Generalized SSSP

- Consider a graph G={V,E} such that in addition to cost c(u,v) for edge e={u,v}, there is vertex cost VC(u) for each vertex. The cost of path is defined as sum of edge costs plus the costs of all vertices on the path (including end points). Write a program that given minimum cost from a given source vertex s to all other vertices.
- Input:
  - Argument 1: file containing graph details.
    - First line `<N> <M>`
    - Next N lines: `a c`, where `c` is cost of vertex `a`.
    - Next M lines define edge cost for each edge
  - Argument 2: source vertex `s` e.g. `2`
- Output: cost of each vertex from source vertex `s`.

Q08: Single Destination SSSP

- Consider a directed graph G={V,E} find the cost to a destination node from each node of the graph. The cost of path is defined as sum of edge costs. Input:
  - Argument 1: file containing graph details.
    - First line `<N> <M>`
    - Next N lines: `a c`, where `c` is cost of vertex `a`.
    - Next M lines define edge cost for each edge
  - Argument 2: destination vertex `d` e.g. `2`
- Output: cost of destination vertex `d` from each vertex.

Q09: Bridge Crossing Problem

- Consider that `n` people need to cross a bridge in the night time with their individual crossing time as $t_1 \leq t_2 \leq ... \leq t_n$, and there is only `1` torch available. A max of 2 people can cross the bridge at one time, and must have torch with them. Thus, the pair takes the time corresponding to slow person in the pair. One person needs to come back with torch so that remaining persons can cross the bridge.
- Design a greedy algorithm for this problem and compute the time it will take for all people to cross the bridge. The input comma separated values of crossing time, e.g.
  - `1, 2, 4, 7, 11`
- Compare the output of your greedy algorithm and answer if greedy approach provides optimal solution.
  - Note: optimal solution for `1,2,5,10` is `17`.

Q10: Kruskal Algo

- Given an input graph (as specified by input file), construct a Minimum Cost Spanning Tree using Kruskal algorithm with using Union-Find approach with Union taking `O(1)` time. Assume that input file has edges defined in ascending (non-decreasing) order of cost.
- Now consider that cost of each edge is increased by `1` and build a new minimum cost spanning tree.

- Provide example of two different graphs, such that in one case two MCSTs are same in another case two MCSTs are different.

Q11: Kruskal Algo

- Given an input graph (as specified by input file), construct a Minimum Cost Spanning Tree using Kruskal algorithm with using Union-Find approach with Find taking `O(1)` time. Assume that input file has edges defined in ascending (non-decreasing) order of cost.
- Now consider that cost of each edge is increased by `1` and build a new minimum cost spanning tree.

- Provide example of two different graphs, such that in one case two MCSTs are same in another case two MCSTs are different.

Q12: Knapsack Problem

- Consider N items with their respecttive weights as $w_1, w_2, \ldots, w_n$ and respective profits as $p_1, p_2, \ldots, p_n$ to be put in a knapsack of weight M. Your objective to fill the knapsack in such a way that your profit is maximized but with minimum number of items.
- Design a greedy implementation of fractional knapsack to achieve the objectives.
- Input a filename
  - first line contains weight of each item
  - second line contains profit of each item.

Compliance

- Please ensure that your programming filename follows the convention
  - `Asn03G<nn>P<mm>.java/c/cpp/py`
  - where `nn` corresponds to your group number and `mm` corresponds to assignment problem number.
  - The programming file should have proper extension.
- In the past two submissions,this convention has been mostly violated. Any violation will result in penalty.
- Further, please ensure to describe specific challenges w.r.t. your problem rather than keeping the generic descriptions such as "we found it difficult, searched internet, discussed among others".
  - Please describe specific challenges that you faced.