School of Computing and Information Systems
COMP20007 Design of Algorithms
Semester 1, 2020
Sample Exam

Note to Students

The details for submission will be made available closer to the exam deadline. This sample exam has questions for 60 marks whereas the actual exam will have 70 marks. The questions in the actual exam will also focus on providing a rationale for your answers. Without a sufficient explanation a correct answer will get 0 marks.

Question 1 [10 Marks]

(a) Perform mergesort on the array $[25, 23, 17, 29, 7, 16, 33, 21]$, showing the result after each recursive call. Indicate which range of elements are being considered during each recursive call by outlining those elements.

[2 marks]

(b) Which two operations must a queue implement? Give the name and a brief description of these two operations.

[2 marks]

(c) Describe how you would implement a queue using a singly linked list to ensure that both of these operations run in $O(1)$ time.

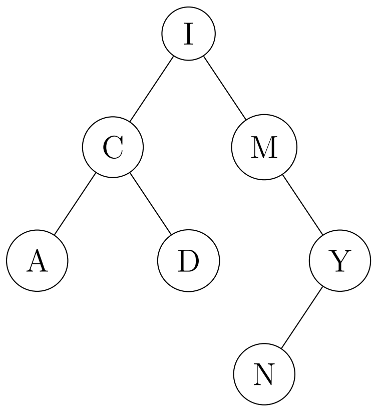
[1 mark]

(d) Insert the characters `C, O, M, P, L, E, X, I, T, Y` into an initially empty 2-3 tree. You must show the state of the 2-3 tree after every insertion.

[2 marks]

(e) Describe which two rotations must be done to balance the following tree? Give the node and the direction of the rotation (left or right).

[1 mark]



(f) Describe how performing a right rotation of the tree above would change the in-order traversal of the tree. You should give the resultant in-order traversal as part of your answer.

[2 marks]

Solutions

(a) Going down the page we split the array until we have arrays of size less than 2. We sort these smaller arrays and then merge them pairwise until we have a single sorted array.

$$[25, 23, 17, 29, 7, 16, 33, 21]$$
$$[25, 23, 17, 29] \quad [7, 16, 33, 21]$$
$$[25, 23] \quad [17, 29] \quad [7, 16] \quad [33, 21]$$
$$[23, 25] \quad [17, 29] \quad [7, 16] \quad [21, 33]$$
$$[17, 23, 25, 29] \quad [7, 16, 21, 33]$$
$$[7, 16, 17, 21, 23, 25, 29, 33]$$

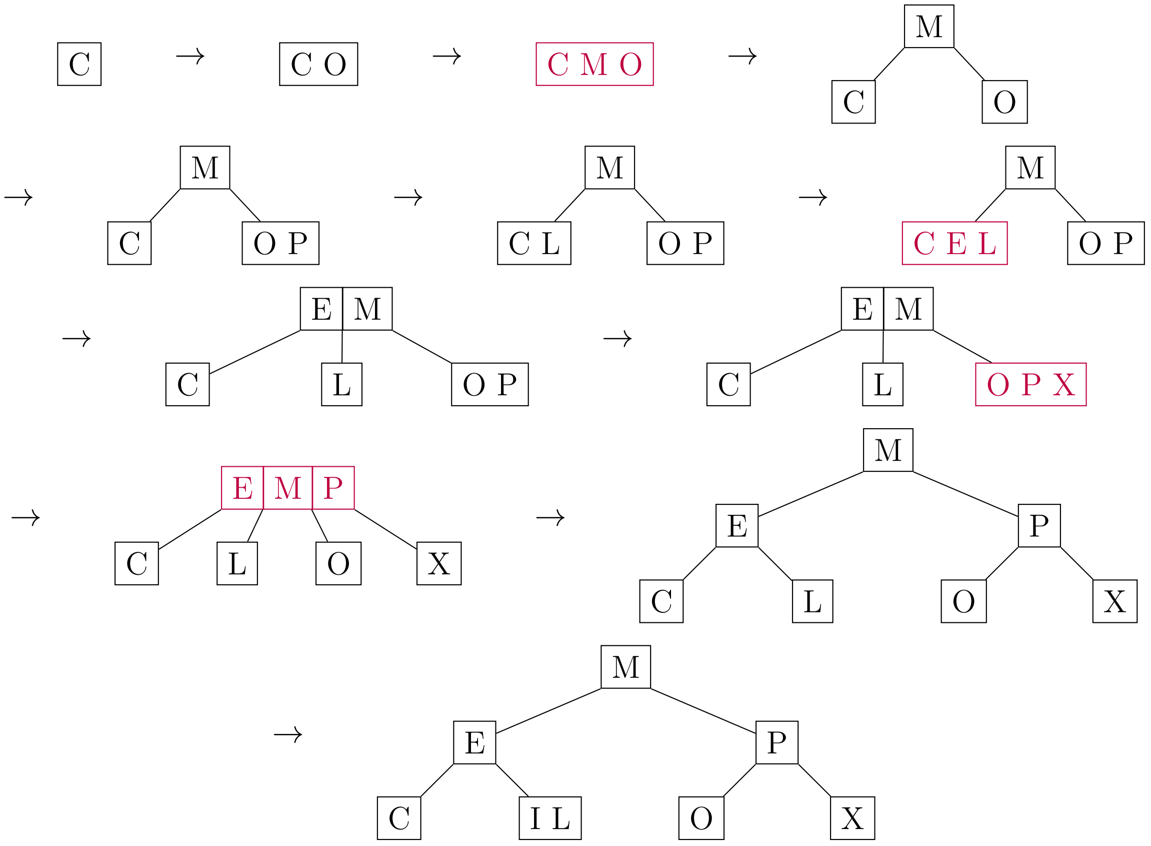
(b) **Enqueue** inserts an element to the "back" of a queue. **Dequeue** removes and returns an element from the "front" (*i.e.,* the element which was inserted into the queue the earliest).

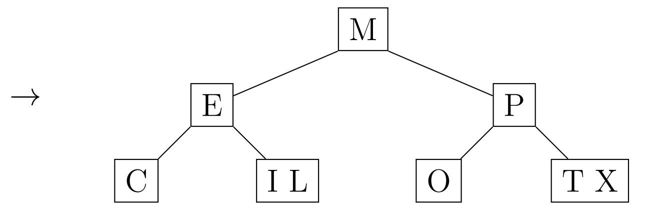
(c) A singly-linked list allows us to insert at both ends in $O(1)$ time, remove the head in $O(1)$ time, and remove from the tail in $O(n)$ time. To achieve $O(1)$ time for both operations:

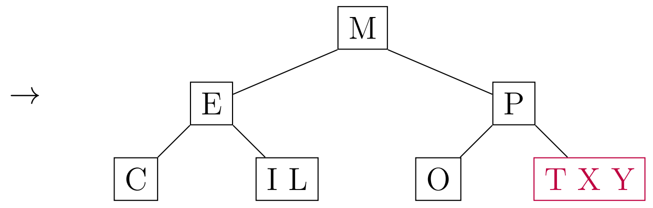
Enqueue: insert the element to the tail of the list.

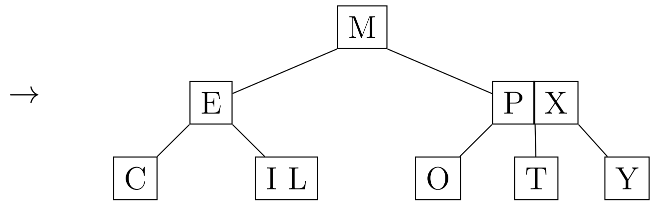
Dequeue: remove and return the element from the head of the list.

(d) The intermediate states of the 2-3 tree (when nodes are over-full and before the problem is fixed) are shown in purple.









(e) The two rotations that must be performed are **rotate Y right** (so that N becomes the parent of Y and the right child of M) and then **rotate M left** (so that I's left child becomes N, with left and right child M and Y resp.).

(f) Performing a rotation on a binary tree does not change the in-order traversal. So the in-order traversal must be A, C, D, I, M, N, Y.

Question 2 [9 Marks]

(a) Demonstrate how to search for the pattern `EAR` in the text `STRINGSEARCH` using Horspool's algorithm. How many comparisons were made?

[1 mark]

(b) A pair of strings are *anagrams* of each other if they contain the exact same letters, for example `"DESSERTS"` and `"STRESSED"` are anagrams.

Write an algorithm which takes as input two strings, S and T, and determines whether or not they are anagrams. Your algorithm should run in $O(n)$ time where $n := \max\{|S|, |T|\}$. You may assume that all characters in S and T are uppercase alphabetic characters.

[4 marks]

(c) Write an algorithm which takes as input an array of n strings, of length no longer than m characters long, and outputs the largest set of strings which are all anagrams of each other.

For example, given the array `[ "ABC", "ABBA", "CAB", "BABA", "CBA", "BAC", "BAD" ]` your algorithm would output `"ABC", "CAB", "CBA", "BAC"` (not necessarily in that order).

Your algorithm must run in $O(mn \log n)$ time. You may again assume that all characters are uppercase alphabetic characters.

[4 marks]

Solutions

(a) Let `-` indicate a successful comparison, and `x` indicate an unsuccessful comparison.

```
    STRINGSEARCH    -->    STRINGSEARCH    -->    STRINGSEARCH
    EAR                      EAR                        EAR
     x-                        x                          x

-->   STRINGSEARCH
             EAR
             ---
```

So there were 7 comparisons made.

(b)
```
function ANAGRAMS($S, T$)
    if LENGTH($S$) $\neq$ LENGTH($T$) then
        return FALSE
    $S' \leftarrow$ COUNTINGSORT($S$)
    $T' \leftarrow$ COUNTINGSORT($T$)
    for $i$ in $0 \ldots |S'| - 1$ do
        if $S'[i] \neq T'[i]$ then
            return FALSE
    return TRUE
```

(c)
```
function AnagramSet($A[0, \dots, n-1]$)
    // Sort each string in alphabetic order. Takes $O(nm)$ time.
    $A' \leftarrow$ copy of $A$
    for $i$ in $0 \dots n-1$ do
        $A'[i] \leftarrow$ CountingSort($A'[i]$)

    // Sort the array of strings lexicographically so that all anagrams appear next to
    // each other in the new array $A''$.
    // Each comparison between strings takes $O(m)$ time so MergeSort
    // will take $O(nm \log n)$ time.
    $A'' \leftarrow$ MergeSort($A'$)

    longestSetIdx $\leftarrow 0$
    currentSetIdx $\leftarrow 0$
    longestSetSize $\leftarrow 1$
    currentSetSize $\leftarrow 1$

    for $i$ in $1 \dots n-1$ do
        if Anagrams($A''[i-1], A''[i]$) then
            currentSetSize $\leftarrow$ currentSetSize $+ 1$
        else
            currentSetIdx $\leftarrow i$
            currentSetSize $\leftarrow 1$

        // Update the longest set if we've found the new longest set so far.
        if currentSetSize $>$ longestSetSize then
            longestSetIdx $\leftarrow$ currentSetIdx
            longestSetSize $\leftarrow$ currentSetSize

    // Now we know what the largest set of anagrams is, output the strings
    for $i$ in $0 \dots$ longestSetSize $- 1$ do
        Output the original string for $A''[$longestSetIdx $+ i]$
```

Question 3 [8 Marks]

(a) For each of the following, explain whether $f(n) \in O(g(n))$, $f(n) \in \Omega(g(n))$, or $f(n) \in \Theta(g(n))$. Only solutions with sufficient justification will be awarded marks.

(i) $f(n) = \log(n)$ and $g(n) = \log(\log(n))$

(ii) $f(n) = \log_3(n)$ and $g(n) = \log_2(n^2)$

(iii) $f(n) = 100\sqrt{n} + 0.01n^2 + 0.001n^3$ and $g(n) = n$

(iv) $f(n) = \log n + n^2$ and $g(n) = n$

[4 marks]

(b) Solve, using the method of repeated substitutions, the following recurrence relation. You must give a closed form expression and a Big-Theta bound for $T(n)$.

$$T(n) = 2T(n-1) + 7, \quad T(2) = 0$$

[2 marks]

(c) Consider the following sorting algorithm:

```
function ChunkSort($A[0 \ldots n-1]$)
    if $n == 2$ and $A[0] > A[1]$ then
        swap $A[0]$ and $A[1]$
    else
        ChunkSort($A[0 \ldots \frac{n}{2}-1]$)
        ChunkSort($A[\frac{n}{2} \ldots n-1]$)
        ChunkSort($A[\frac{n}{4} \ldots \frac{3n}{4}-1]$)
        ChunkSort($A[0 \ldots \frac{n}{2}-1]$)
        ChunkSort($A[\frac{n}{2} \ldots n-1]$)
        ChunkSort($A[\frac{n}{4} \ldots \frac{3n}{4}-1]$)
```

Write the recurrence relation (including the base case) for the number of comparisons of Chunk-Sort.

[1 mark]

(d) Recall, the master theorem states that if T is a recurrence relation such that $T(n) = aT(\frac{n}{b}) + \Theta(n^c)$ and $T(1) =$ constant then,

$$T(n) \in \begin{cases} \Theta\left(n^c\right) & \text{if } c > \log_b(a) \\ \Theta\left(n^c \log n\right) & \text{if } c = \log_b(a) \\ \Theta\left(n^{log_b(a)}\right) & \text{if } c < \log_b(a) \end{cases}.$$

Use the master theorem to find the time complexity of ChunkSort.

[1 mark]

Solutions

(a) (i) Using l'Hopital's rule

$$\lim_{n\to\infty} \frac{\log(n)}{\log(\log(n))} = \lim_{n\to\infty} \frac{\frac{d}{dn}(\log(n))}{\frac{d}{dn}(\log(\log(n)))} = \lim_{n\to\infty} \frac{n^{-1}}{\frac{n^{-1}}{\log(n)}} = \lim_{n\to\infty} \log(n) = \infty$$

So $f(n) \in \Omega(g(n))$.

(ii) $g(n) = \log_2(n^2) = 2\log_2(n) = \frac{2}{\log_3(2)}\log_3(n)$, so $f(n) \in \Theta(g(n))$.

(iii) $f(n) \in \Theta(n^3)$ and $g(n) \in \Theta(n)$ so $f(n) \in \Omega(g(n))$.

(iv) $f(n) \in \Theta(n^2)$ and so $f(n) \in \Omega(g(n))$.

(b)

$$\begin{aligned}
T(n) &= 2T(n-1) + 7 \\
&= 2\big(2T(n-2) + 7\big) + 7 \\
&= 2\Big(2\big(2T(n-3) + 7\big) + 7\Big) + 7 \\
&= 2^3T(n-3) + 2^2 \times 7 + 2^1 \times 7 + 7 \\
&\vdots \\
&= 2^kT(n-k) + 7\sum_{i=0}^{k-1} 2^i \\
&\vdots \qquad \text{let } k = n-2 \\
&= 2^{n-2}T(2) + 7\sum_{i=0}^{n-3} 2^i \\
&= 2^{n-2} \times 0 + 7\left(2^{n-2} - 1\right) \\
&= 7 \times 2^{n-2} - 7 \\
&\in \Theta(2^n)
\end{aligned}$$

(c) The recurrence relation for ChunkSort is

$$T(n) = 6T\left(\frac{n}{2}\right), \quad T(2) = 1$$

(d) Using the Master Theorem, with $a = 6$, $b = 2$ and $c = 0$, we get,

$$T(n) \in \Theta\left(n^{\log_2(6)}\right)$$

Question 4 [10 Marks]

(a) Consider a hash table with 8 slots and a hash function $h(k) = k \mod 8$, which uses open addressing with a step size of 1. Show the contents of the table after inserting 16, 23, 7, 10 and 12.

[2 marks]

(b) How many comparisons are required to lookup the key 7? Indicate which comparisons are performed. You can assume that we can check if a slot is empty without making any comparisons.

[1 mark]

(c) How many comparisons are required to lookup the key 15? Indicate which comparisons are performed. You can assume that we can check if a slot is empty without making any comparisons.

[1 mark]

(d) Consider again a hash table with 8 slots and a hash function $h_1(k) = k \mod 8$. This time, the hash table uses open addressing and double hashing, with the step size determined by the hash function $h_2(k) = k \mod 3 + 1$.

Explain why $h_2(k)$ must contain the $+1$ term?

[1 mark]

(e) Show the state of the hash table after using this double hashing scheme (*i.e.,* the scheme described in Part (d)) to insert 9, 17, 4 and 11 into an empty hash table.

[1 mark]

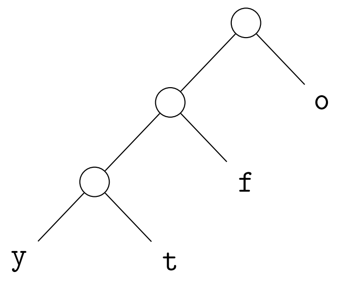
(f) Use Huffman's algorithm to construct a Huffman tree for the string `"free-coffee"`.

[2 marks]

(g) What is the encoded version of this string, using your Huffman tree? Let each left child be 0 and each right child be 1. Give the total length of the encoding in bits.

[1 mark]

(h) Using the following Huffman tree (again, using 0 for left and 1 for right), give the codes for `f`, `o`, `t`, and `y` and decode 0111001000.



[1 mark]

Solutions

(a)

0	1	2	3	4	5	6	7
16	7	10		12			23

(b) Three comparisons are required, at indices 7, 0, and 1.

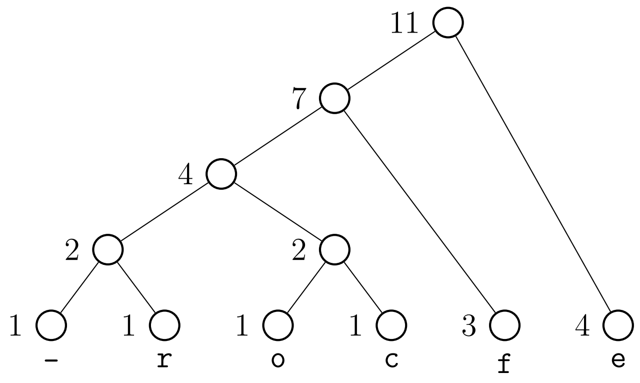
(c) 15 hashes to 7. So indices 7, 0, 1, and 2 are checked (unsuccessfully). Since 3 is empty this process finishes without finding 15 after **4 comparisons**.

(d) If the secondary hash function could map elements to 0, then the skip size would be 0 and the hash table would be unable to deal with collisions for some values. The skip size must be positive, hence the +1.

(e)

0	1	2	3	4	5	6	7
	9		11	17		4	

(f) The frequencies are $\{\texttt{e} : 4, \texttt{f} : 3, \texttt{c} : 1, \texttt{o} : 1, \texttt{r} : 1, - : 1\}$



(g) Let left be 0 and right be 1. The encoded version of "`free-coffee`" is then

```
01 0001 1 1 0000 0011 0010 01 01 1 1
```

with a length of 26.

(h) The decoded version of 0111001000 is `footy`.

Question 5 [10 Marks]

(a) A string of parentheses (`"("` and `")"`) is *valid* if no pair of parentheses is "closed" before it is "opened", `"(())()"` is valid while `"())("` is not.

The 5 valid strings of 6 parentheses are:

`"()()()"`, `"(())()"`, `"()(())"`, `"(()())"` and `"((()))"`.

Give 3 (three) examples of valid strings of parentheses containing 8 (eight) parentheses.

[1 mark]

(b) Write a recursive formula and base case(s) for the subproblem N_i, where N_i is defined as the number of different valid strings containing exactly i parentheses.

[4 marks]

(c) Using the recursive formula you derived in Part (b) write pseudocode for an algorithm which computes the number of different valid strings containing n parentheses.

[3 marks]

(d) What is the space complexity of your algorithm? Give an answer in Big-Theta notation and justify your answer.

[1 mark]

(e) What is the time complexity of your algorithm? Give an answer in Big-Theta notation and justify your answer.

[1 mark]

Solutions

(a) Some examples are `"()()()()"`, `"(((())))"`, and `"(())()()"`.

(b) We can determine the number of valid strings there are by considering the following scenario: place one set of parentheses and then determine how many ways we can fill the gaps.

For example for n parentheses, we've already placed 2 parentheses, so we will put some even number within these (lets say k) and the remaining parentheses after the pair we've already placed:

$$(\underbrace{\qquad\quad}_{N_k \text{ ways}})\underbrace{\qquad\quad}_{N_{n-2-k} \text{ ways}}$$

The recursive formula for N_n is then given by,

$$N_0 = 1$$
$$N_n = \begin{cases} 0 & \text{if } n \text{ odd} \\ \sum_{i\in\{0,2,4,\ldots,n-2\}} N_i \times N_{n-2-i} & \text{if } n \text{ even} \end{cases}$$

(c)
```
function PARENTHESES($n$)
    if $n$ % $2 == 1$ then
        return 0
    $N \leftarrow$ array with $n + 1$ elements
    $N[0] \leftarrow 1$
    for $k$ in $2, 4, \ldots, n$ do
        $N[k] = 0$
        for $i$ in $0, 2, \ldots, n - 2$ do
            $N[k] \leftarrow N[k] + N[i] \times N[n - 2 - i]$
    return $N[n]$
```

(d) The algorithm requires an additional array with $n + 1$ elements, so the space complexity is $\Theta(n)$.

(e) To determine the time complexity of this dynamic programming algorithm we first note that there are $n/2 = \Theta(n)$ subproblems to solve, and each one requires a loop over up to $n/2 = \Theta(n)$ elements. So the runtime complexity of the algorithm is $\Theta(n^2)$.

Question 6 [13 Marks]

(a) Show how we interpret the array $\left[\texttt{null}, 8, 4, 3, 5, 2, 13, 6, 4\right]$ as a tree with the indexing scheme used for heaps.

[1 mark]

(b) Run the bottom-up construct heap algorithm to convert this array into a *max-heap*. Show the state of the heap (in tree form) after each step, indicating exactly which operations are being performed.

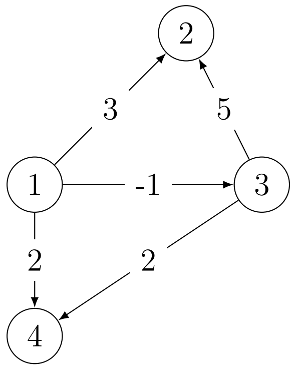
[2 marks]

(c) Now show the state of the heap after one RemoveMax operation. Which element is returned after this operation is performed?

[1 mark]

(d) Give the weights matrix for the following graph.

[1 mark]



(e) Run Floyd's algorithm to find the shortest paths between each pair of vertices in the graph from part d. Write the matrices after each step of the algorithm

[3 marks]

(f) Describe what the time complexity of Floyd's algorithm is (assume the graph has n vertices and m edges).

[1 mark]

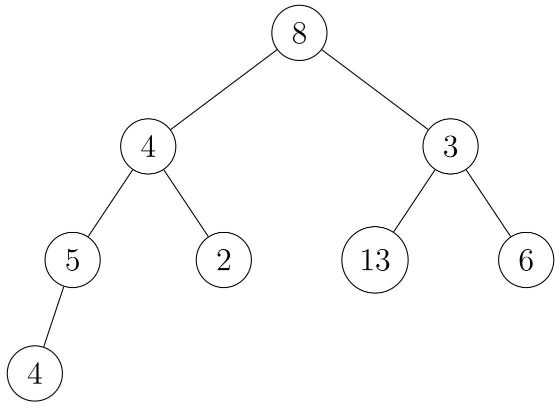
(g) Using Dijkstra(G, u, v) as a helper function which returns the cost of the shortest path between u and v in a graph G, write an algorithm in pseudocode for computing the *all pairs shortest paths* in a graph G with non-negative edge weights. The graph G has n vertices and m edges and is sparse, *i.e.,* $m \in O(n)$.

Your algorithm must have a time complexity better (*i.e.,* smaller) than Floyd's algorithm.

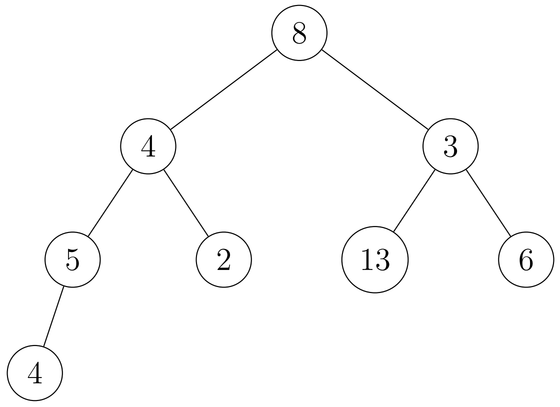
[3 marks]

Solutions

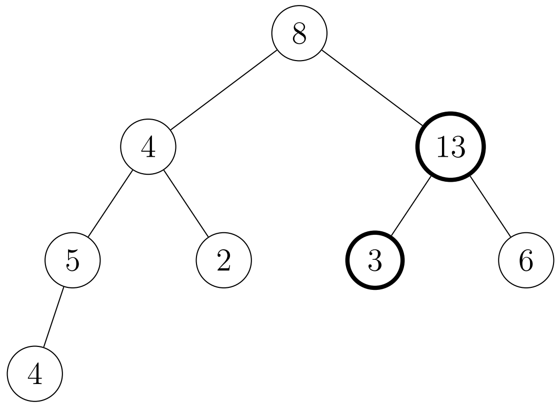
(a)



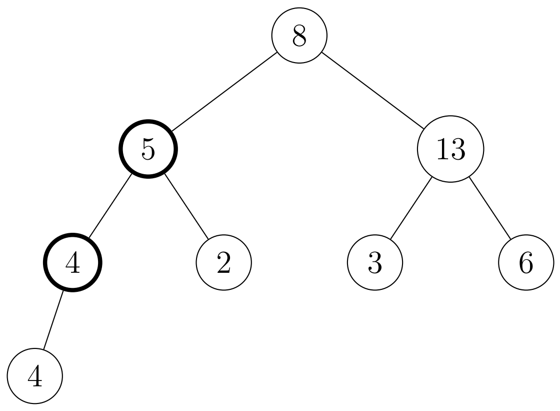
(b) Bold elements indicate nodes whose values have just been swapped. First we sift down 5 (nothing to be done).



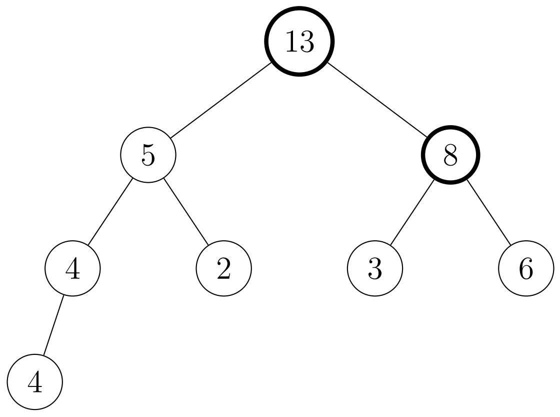
Now we sift down the 3 in the second layer.



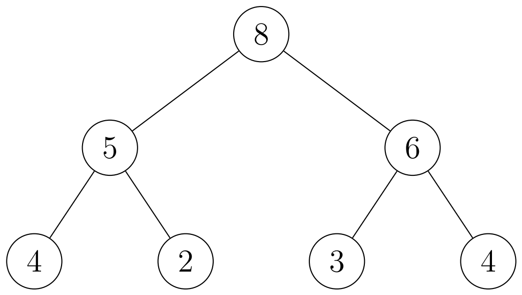
Then the 4 in the second layer is sifted down.



Then finally the root is sifted down.



(d) Element 13 is removed by the RemoveMax operation. The state of the heap after this operation is:



(d) The weights matrix is given by

$$\begin{bmatrix} 0 & 3 & -1 & 2 \\ \infty & 0 & \infty & \infty \\ \infty & 5 & 0 & 2 \\ \infty & \infty & \infty & 0 \end{bmatrix}$$

(e) Running Floyd's algorithm yields the following matrices. The row and column in question will be appear blue, while any elements considered for updating will appear in red.

$$W^0 = \begin{bmatrix} 0 & 3 & -1 & 2 \\ \infty & 0 & \infty & \infty \\ \infty & 5 & 0 & 2 \\ \infty & \infty & \infty & 0 \end{bmatrix}$$

$$W^1 = \begin{bmatrix} 0 & 3 & -1 & 2 \\ \infty & 0 & \infty & \infty \\ \infty & 5 & 0 & 2 \\ \infty & \infty & \infty & 0 \end{bmatrix}$$

$$W^2 = \begin{bmatrix} 0 & 3 & -1 & 2 \\ \infty & 0 & \infty & \infty \\ \infty & 5 & 0 & 2 \\ \infty & \infty & \infty & 0 \end{bmatrix}$$

$$W^3 = \begin{bmatrix} 0 & 3 & -1 & 1 \\ \infty & 0 & \infty & \infty \\ \infty & 5 & 0 & 2 \\ \infty & \infty & \infty & 0 \end{bmatrix}$$

$$W^4 = \begin{bmatrix} 0 & 3 & -1 & 1 \\ \infty & 0 & \infty & \infty \\ \infty & 5 & 0 & 2 \\ \infty & \infty & \infty & 0 \end{bmatrix}$$

(f) Floyd's algorithm is a dynamic programming algorithm which solves n subproblems (the intermediate weights matrices), each of which requiring up to $\Theta(n^2)$ updates. So Floyd's algorithm is $\Theta(n^3)$.

(g) The following algorithm will run in $O(n^2 \log n)$ time given the graph is sparse, because in a sparse graph Dijkstra's algorithm takes $O((n + m) \log n) = O((n + n) \log n) = O(n \log n)$ time, and we run this as a helper function n times.

```
function MyAPSP($G = (V, E)$)
    Paths ← $n \times n$ matrix
    for $u$ in $V$ do
        $u$Paths ← Dijkstras($G, u$)
        for $v$ in $V$ do
            Paths[$u$][$v$] ← $u$Paths[$v$]
    return Paths
```

END OF TEST