1. [5 pts] In a max-heap where all elements are distinct, where is the smallest element located? Please explain.

The smallest elements must be located in one of the leaves of this heap, which means that the node that contains the smallest element can't have any sub-nodes or sub-child.
proof: if we assume that the smallest element located in a node which is not a leaf of the tree, so it will have one or two sub nodes, and the rule of the max-heap is the node's element must larger than its sub nodes element, so the smallest element must larger than its sub node element — Contradiction! so the smallest element must be the leaf of this tree.

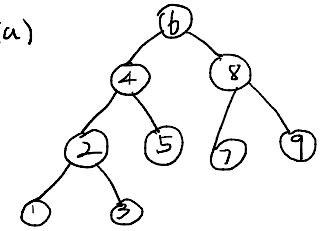
2. [5pts] Ms RB would like to transform a red-black tree into a black-only tree by setting the children of the red nodes to become the children of the parent of the red nodes, and then removing the red nodes. What can you say about the leaves' depth of the resulting tree? Please explain.

(original RBT is T)
Now let the new black-only tree be T', and the height of the T' is h' (original RBT height is h).
Because we are only removing the red node so the number of leaves will not decrease, namely, #leaves in T = #leaves in T'.
Since T is a full binary tree (every internal node will have two children), so #leaves in T = n+1 (n is the internal nodes),
Since each node in T' will have at least 2 childrens and at most 4 children, so #leaves in T' $\in [2^{h'}, 4^{h'}]$. And #leaves in T = #leaves in T', therefore $2^{h'} \le n+1 \le 4^{h'}$

$\frac{\log(n+1)}{2} \le h' \le \log(n+1)$ And $h' \ge \frac{h}{2}$

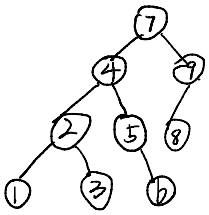
3. [10pts] Suppose you need to represent a set of data whose keys are: 1, 2, 3, 4, 5, 6, 7, 8, 9 as an AVL tree.
   (a) [5pts] Draw an instantiation of the shallowest AVL tree that you can construct for the above data set and explain why it is indeed the shallowest AVL tree possible for the data set.
   (b) [5pts] Draw an instantiation of the deepest AVL tree that you can construct for the above data set and explain why it is indeed the deepest AVL tree possible for the data set.

(a)



In Order to draw a shallowest AVL tree, we must make the tree as full as possible and a tree node will have 2 children at most, therefore we will try to make a full binary tree.

(b)



In Order to construct a deepest AVL Tree. We can define N_h as the minimum number of nodes to build a AVL Tree with height h. So we have $N_h = N_{h-1} + N_{h-2} + 1$

$N_0 = 1$
$N_1 = 2$
$N_2 = 4$
$N_3 = 7$
$N_4 = 12$

⇒ To build a AVL Tree with height 4, we must have at least 12 nodes but we only have 9 nodes, so the height of the deepest AVL tree that can be built is 3.

4. [5 pts] Mr H argues that for a hash table with chaining and simple ~~universal~~ uniform hashing, the expected time complexity to search for an item can be made to be $O(\frac{1}{n})$. He suggested that this time complexity is achievable simply by setting the number of slots in the hash table to be n^2 (n is the number of elements in the table) because this setting will lead to a load factor $\alpha = \frac{1}{n}$. Is Mr H correct? Please explain why or why not.

NO, It is not correct

The probability of an element be assigned in slot $j \in [0, n^2-1]$ is $\frac{1}{n^2}$ because it is simple uniform hashing, and let X_{ij} be $I\{h(k_i) = h(k_j)\}$
So the expected number of being checked is $E\left[\frac{1}{n}\sum_{i=1}^{n}(1+\sum_{j=i+1}^{n}X_{ij})\right]$

$$E\left[\frac{1}{n}\sum_{i=1}^{n}(1+\sum_{j=i+1}^{n}X_{ij})\right] = \frac{1}{n}\sum_{i=1}^{n}(1+\sum_{j=i+1}^{n}E[X_{ij}]) \quad \because E[X_{ij}] = \frac{1}{n^2}$$

$$= \frac{1}{n}\sum_{i=1}^{n}(1+\sum_{j=i+1}^{n}\frac{1}{n^2}) = \frac{1}{n}\sum_{i=1}^{n}(1+\frac{n-i}{n^2}) = 1+\frac{1}{n}\cdot\frac{1}{n^2}\sum_{i=1}^{n}(n-i) = 1+\frac{1}{n^3}\left(n^2-\frac{(1+n)n}{2}\right) = 1+\frac{1}{2n}-\frac{1}{2n^2}$$

$$\lim_{n\to\infty}\frac{1+\frac{1}{2n}-\frac{1}{2n^2}}{\frac{1}{n}} = \lim_{n\to\infty} n+\frac{1}{2}-\frac{1}{2n} = \infty$$

if the time complexity is $O(\frac{1}{n})$, the limit must be 0 or c, however, it is ∞ so it should belong to $\Omega(\frac{1}{n})$

5. [5 pts] Mr B said that he found a comparison-based method to construct a Binary Search Tree in $O(n)$, where n is the number of elements in the data set. Could this be true? Please provide a proof or counter-example to support your argument.

False

If we using the ordinary method to build a BST, the time complexity would be $O(n \log h)$ (h is the height of Tree) Because We will insert the elements one by one into the tree. In order to build a tree in $O(n)$, We can first sort the tree and determine the location of the element in the tree in advanced, therefore We have to find a comparison based method which sort an array in $O(n)$ and We have to assume that we construct the tree with sorted array in $O(n)$.

For the second one, We can build the tree by extracting the median as a node, and then do that again on the subarrays that are array$[s, m)$ and array$(m, e]$ until there is only 1 element in the array. as We are choosing the median of the array to build a node and all the elements in the array will be the nodes, So we will do the recursion on n time, the time complexity is $O(n)$.

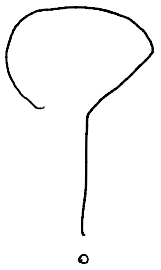
However, finding a comparison-based method of sorting an array in $O(n)$ is impossible, the best performance of the comparison-based method is $\theta(n \log n)$ and if we using the counting sort, we can built it $O(n)$.

6. [10 pts] Consider a hash table T with m slots. Suppose T contains a single element, and this element has key k. Ms Search has been searching for r keys that are different from k in hash table T. Assuming T uses simple uniform hashing and chaining, what is the probability that at least one of the r searches collide with the m slot containing key k? Please provide the derivation.

For any $i \in U$, $P(h(k) = h(i)) = \frac{1}{m}$ Because it is a simple uniform hashing, And the probability of at least one key collides with m slot = 1- probability of none of r keys collide with the m slot.

$P(\text{none of the r keys collide to m}) = \frac{1}{m}\left(\frac{m-1}{m}\right)^r$

So the probability of at least one r search collide with m slot is $1 - \frac{1}{m}\left(\frac{m-1}{m}\right)^r$



7. [10 pts] Recall the matrix chain multiplication problem we discussed in class (week-8 Tuesday, based on [CLRS] 15.2). Mr SP says that we can compute the optimal solution to this problem faster, simply by recursively splitting the (sub-)chain of matrices $A_s \cdot A_{s+1} \cdots A_e$ at A_k, where $1 \le s \le e$, the size of A_i for $i \in [s, e]$ is $p_{i-1} \times p_i$, and k is selected to minimise the value $p_{i-1} \times p_k \times p_j$ (i.e., $k = \arg\min_{i \in [s, e-1]} p_i$). To make it clearer, below is the pseudo-code of the algorithm Mr SP proposed. The output of the pseudo-code is a string of the chain of matrices together with their parenthesis.

Algorithm 1 SP-MatrixChainMult(A chain of matrices $A_s, A_{s+1}, \cdots, A_e$)

1: **if** $s == e$ **then**
2: Return "A_s"
3: Let $k = \arg\min_{i \in [s, e-1]} p_i$
4: Return "(SP-MatrixChainMult($A_s, \cdots A_k$)) · (SP-MatrixChainMult($A_{k+1}, \cdots A_e$))"

Incorrect

Supposed that we have a chain of matrices with length 6 $\begin{pmatrix} A_1 & A_2 & A_3 & A_4 & A_5 & A_6 \\ 30\times35 & 35\times15 & 15\times5 & 5\times10 & 10\times20 & 20\times25 \end{pmatrix}$, and if we use the Mr SP Algorithem We will receive $\left[((A_1 \cdot A_2) \cdot A_3) \cdot (A_4 \cdot (A_5 \cdot A_6))\right]$ because $(30\times25\times k_1=5)$, $(30\times5\times k_2=15)$, $(5\times25\times k_3=10)$ is minimum, but if we use the DP Algorithem in CLRS, We will receive $\left[(A_1 \cdot (A_2 \cdot A_3)) \cdot ((A_4 \cdot A_5) \cdot A_6)\right]$.

The cost of the Mr SP algorithem: $30\times35\times15 + 30\times15\times5 + 10\times20\times25 + 10\times25\times5 + 30\times5\times25 = 28000$ (15750, 2250, 5000, 1250, 3750)

The cost of the CLRS DP algorithem: 15125

$T(SP) = 28000 > T(DP) = 15125$

So it is not the minimum

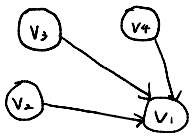
(reference: This matrices chain is taken from CLRS 15.2)

8. [10 pts] As a program manager at a large company, Mr PM is overseeing multiple projects. As a good program manager, he needs to foresee whether there will be delays in each of his projects. To this end, he asked your help to develop a Dynamic-Programming approach to compute an estimated longest duration to finish a project.

To compute the above estimate, you can assume a project is divided into multiple tasks, starting from an initial task and ending at a final task. The dependencies between tasks can be represented as a weighted directed graph, called the task graph. The vertices of a task graph represent the tasks, an edge from a vertex v to another vertex v' in a task graph means the task represented by v must be completed first before the task represented by v' can start, and the weight associated with with vertex $\overline{vv'}$ indicates the time duration to complete the task represented by v. One can then compute the estimated longest duration to finish a project by finding the longest path from the vertex representing the initial task to the vertex representing the final task in the task graph.

In this question, you need to provide the sub-problems and recursive definition of the optimal value function (i.e., step-1 and step-2 of Dynamic Programming development steps, as discussed in class in week8 Tuesday lecture based on [CLRS] 15.Intro and 15.3) to find the longest path in the task graph.

Assumption ①: if there is a vertex V_1 that has more than one vertices $V_{k(k\geq 2)}$ have edge connected to V_1, then any of the V_K that is finished can start V_1.



(V_1 task starts after V_2 or V_3 or V_4 is finished)

Assumption②: The Task Graph is a DAG because the circulation in the Task Graph is meaningless and violate the laws of the real world, you don't need to start the task that is already finished. The most crucial reason is that if there is a circulation in the Graph, then the path will walk through that circulation again and again, the longest duration will be ∞, and it is meaningless.

Step 1 Sub-problems: length of the longest subpaths from the initial task $d(s,v)$ $v \in G.V$

step 2 Relation between sub graph: $d(s,e) = \max_{(u,e)\in G.E} (d(s,u) + w(u,e))$

9.

(a) [10 pts] Please provide step-1 and step-2 of the dynamic programming development steps (the steps are as discussed in class in week8 Tuesday lecture based on [CLRS] 15.Intro and 15.3).

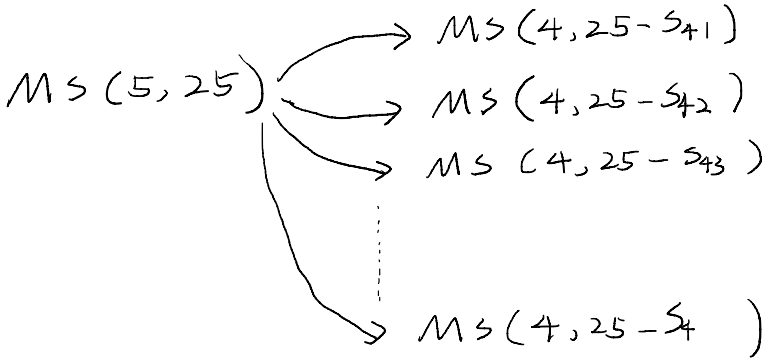
step 1 Sub-problems: let TB = Total Budget = B×#10000, b = budget, function $MS(n,b)$ return maximum salary

$MS(n,b)$: In the first n positions, the maximum total salary that MsC can pay for with budget b

step 2 Relation between sub-problems: S_{nj} means at position n, the salary requested by the j^{th} applicants

$MS(n,b) = \max\{MS(n-1, b-S_{nj}) + S_{nj}\}$ where $S_{nj} < b$, $1 \leq j \leq K_n$

(b) [4 pts] Please show that for the sub-problems you have defined in Q9, the sub-problem graph is a directed acyclic graph.



$MS(n,b)$ will have K_n children, and each children will have K_{n-1} children, so at the end, it will reach $MS(0,\cdots)$. Therefore, the sub problem graph will be a Tree graph where each node will have multiple children, the tree graph can never have a circulation. So it must be DAG.

We want to solve $MS(n,25)$, We have to figure out all the sub-problems in the set $\{MS(n-1, S_{nj}) \mid j \in [1, K_n]\}$, so the edges will be created from $MS(n,25)$ to its sub problems. As the n will only decrease, so it will never have chance to form a circulation.

(c) [6 pts] Please provide the pseudo-code for top-down and bottom-up dynamic programming to help Ms C.

Top-down

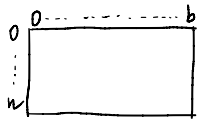
```
Maximum-Salary ( int n , int b , MEMO )
    if ( MEMO[n] [b] != null )
        return MEMO[n] [b];
    int salary = 0;
    if (n=0 or b=0)
        return salary;
    else
        salary = Max { for j=1 to kn : if (b-snj >0) then Maximum-Salary (n-1, b-snj) + snj } ;
    MEMO[n][b] = salary
    return salary
```

Bottom-up

```
Maximum-Salary (int n, int b, MEMO)
    initialize the MEMO to let ∀i,j MEMO[0][j], MEMO[j][0] = 0
    Create new array applicants[n+1] [b+1]
    for i = 1 to n
        for j = 1 to b
            for K = 1 to ki
                if (j - sik > 0 and MEMO[i-1][j-sik] ≠ 0)
                    MEMO[i] [j] = Max( MEMO[i][j] , MEMO[i-1] [j - sik] + sik)
                    Applicant_selectAt = k+1
                    parent = j - sik
                else
                    MEMO[i] [j] = 0;
            pair<int,int> = Selected_Applicant (Applicant_selectedAt, parent)
            Applicants[i][j] = Selected_Applicant
    return MEMO[n][b] and Applicants

Find_Applicant (Vector<vector<pair<int, int>>> Applicants , int n , int b )
    int parent = Applicants[n][b].second
    int chosen_one = Applicants[n][b].first
    Creat new array Answer with length n
    Answer[n-1] = chosen_one
    for (int pos = n-1 ; pos>0 ; pos-- )
        answer[pos-1] = Applicants[pos][parent].first
        parent = Applicants[pos][parent].second
    return answer
```

(d) [5] Please derive the asymptotic time complexity of your bottom-up dynamic programming solution.

#subproblems : $\theta(n \cdot b)$ — because we are filling the table 

, each entry of the table is a subproblem. and we have $n \cdot b$ subproblems where n is the # positions, b is the budget.

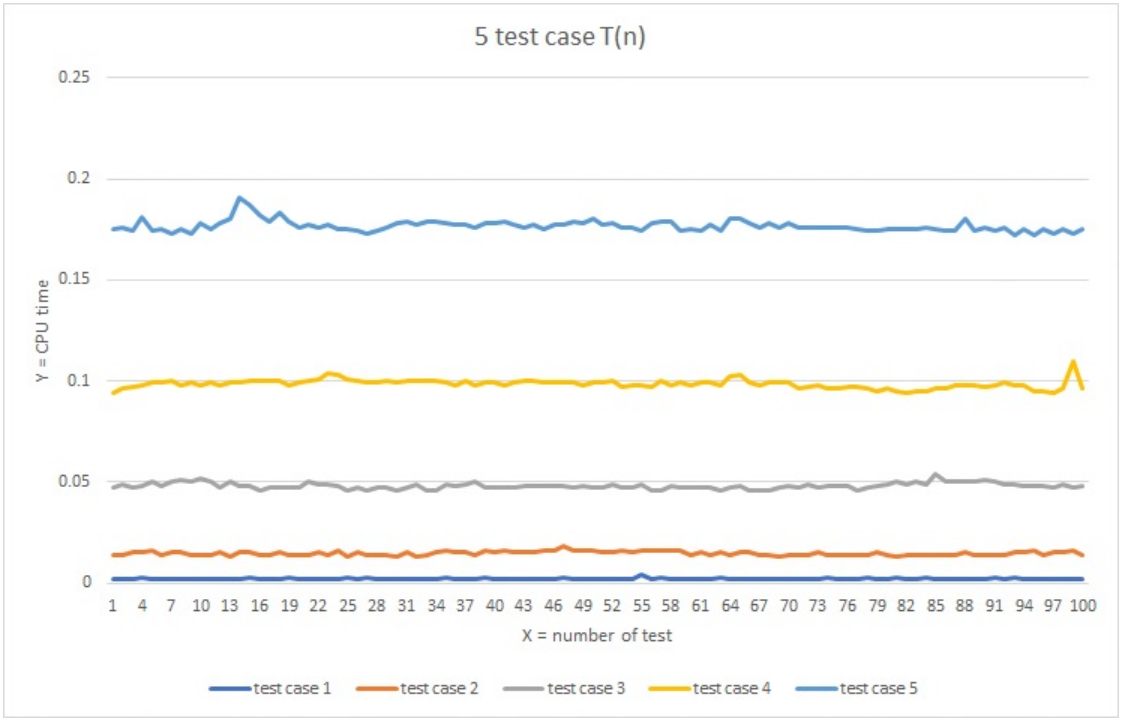
compute time/subproblem: $\theta(Kn \cdot 1)$ — because we have a "for 0 to ki loop" to check whether all the ki applicant's salary can maximize the budget. assume the assignment Time complexity is $\theta(1)$, so we will have $\theta(k_i) \cdot \theta(1)$ for each subproblem

Total time = #sub-problems × time/sub-problem = $\theta(n \cdot b) \cdot \theta(Kn \cdot 1) = \theta(n \cdot b \cdot kn) = \theta(n^3)$

Time for finding the Applicants : $\theta(n)$
So total time will be $\theta(n^3) + \theta(n) = \theta(n^3)$

(f) [5 pts] Please compare the empirical running time of your program and the asymptotic time complexity you have derived in Q9.d. For this purpose, you need to create at least 5 scenarios with increasing parameter size, and provide a comparison between the theoretical and empirical time-complexity.

with the restrition on n, b, kn ($1 \le n \le 25$, $10 \le b \le 300$, $1 \le K_i \le 150$ for $i \in [1,n]$)
In Order to create 5 test cases and make it uniform, I used a factor $nj = \frac{n_{max}}{5}$ $bj = \frac{B_{max}}{5}$ $kj = \frac{k_{max}}{5}$, so $nj = 5$ $bj = 60$ $kj = 30$
therefor the 5 test cases will be:
[n=5, b=70, K=30]
[n=10, b=130, K=60]
[n=15, b=180, K=90]
[n=20, b=230, K=120]
[n=25, b=280, K=150]
for each test case, it will run 100 times to get the average CPU time.



Empirical time
- $Avg(T_1) = 0.002\,s$
- $Avg(T_2) = 0.015\,s$
- $Avg(T_3) = 0.048\,s$
- $Avg(T_4) = 0.098\,s$
- $Avg(T_5) = 0.177\,s$

asymptotic time
- $T(1) = 10500$
- $T(2) = 78000$
- $T(3) = 243000$
- $T(4) = 552000$
- $T(5) = 1050000$

if We analyze it correctly, then $\frac{T_{(1)}}{Avg(T_1)} = \frac{T_{(2)}}{Avg(T_2)} = \frac{T_{(3)}}{Avg(T_3)} = \frac{T_{(4)}}{Avg(T_4)} = \frac{T_{(5)}}{Avg(T_5)}$

$\frac{T(1)}{Avg(T_1)} = 5250000$ simplified as 5.25

$\frac{T(2)}{Avg(T_2)} = 5200000$ simplified as 5.2

$\frac{T(3)}{Avg(T_3)} = 5062500$ simplified as 5.1

$\frac{T(4)}{Avg(T_4)} = 5632653$ simplified as 5.6

$\frac{T(5)}{Avg(T_5)} = 5932203$ simplified as 5.9

We can say $\frac{T_{(1)}}{Avg(T_1)} = \frac{T_{(2)}}{Avg(T_2)} = \frac{T_{(3)}}{Avg(T_3)} = \frac{T_{(4)}}{Avg(T_4)} = \frac{T_{(5)}}{Avg(T_5)} \approx c$ with maximum error 0.7, so the Conclusion of the comparison is that our asymptotic time is correspond to empirical analysis.