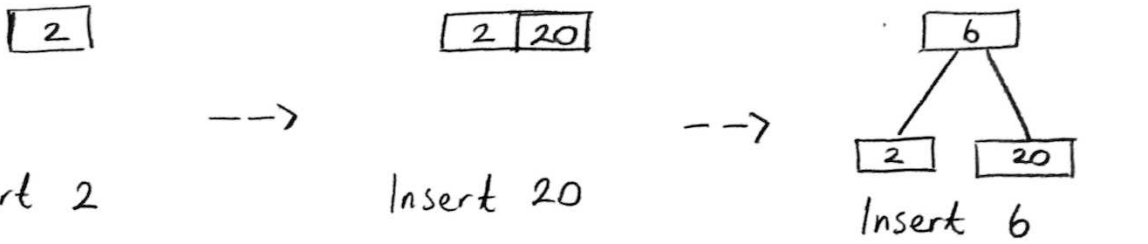
Berk Gakar - 22003021 - CS202-3

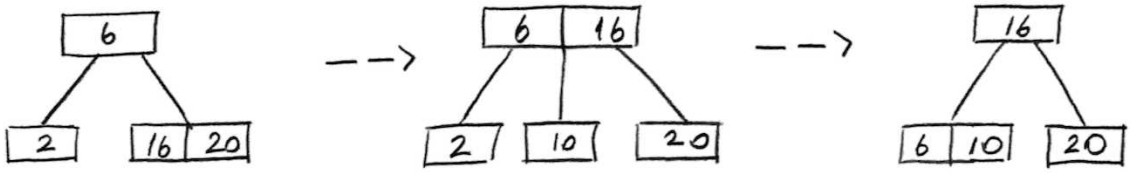
Homework 4

Question 1

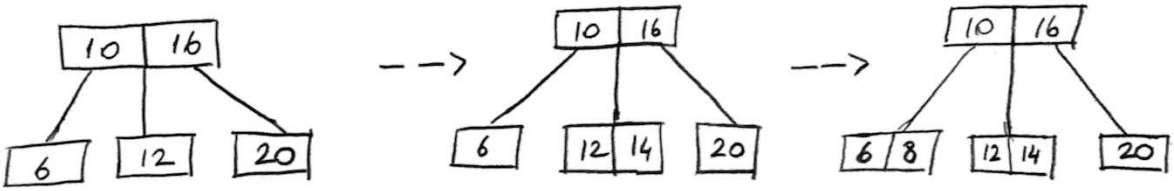
a)



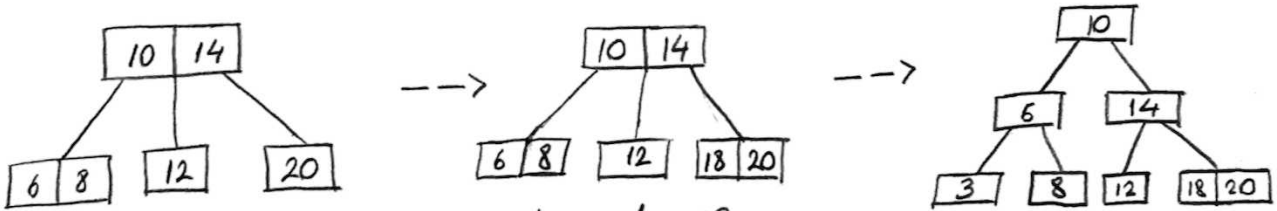
Insert 2 → Insert 20 → Insert 6



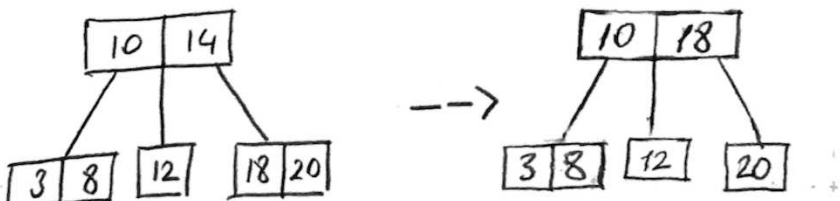
Insert 16 → Insert 10 → Delete 2



Insert 12 → Insert 14 → Insert 8

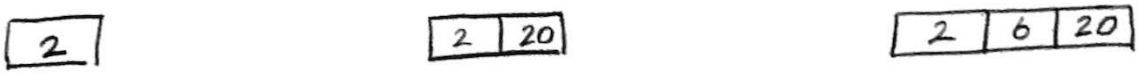


Delete 16 → Insert 18 → Insert 3

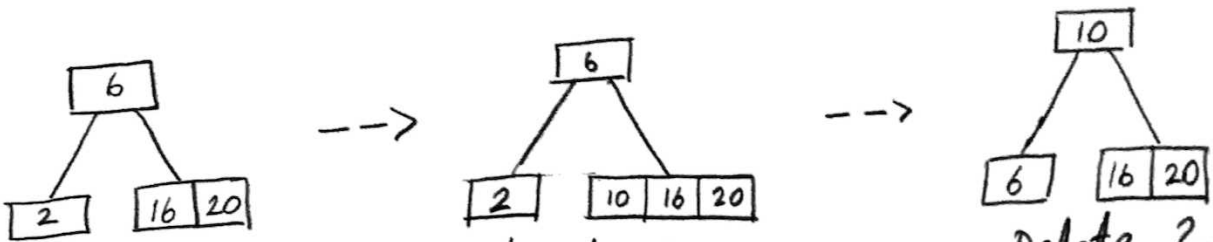


Delete 6 → Delete 14

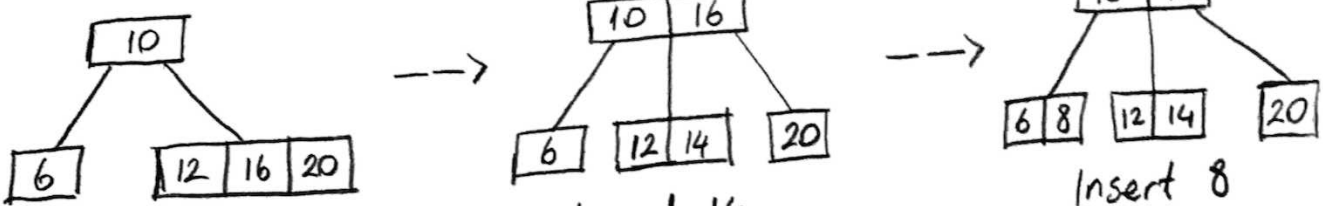
b)



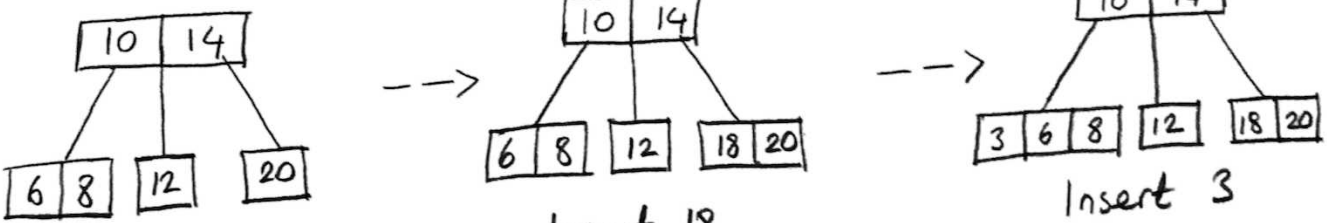
Insert 2 --> Insert 20 --> Insert 6



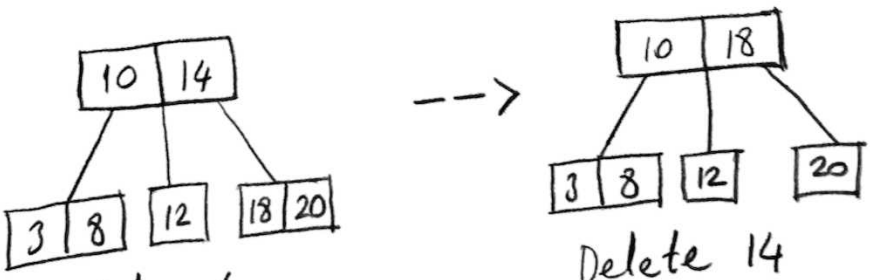
Insert 16 --> Insert 10 --> Delete 2



Insert 12 --> Insert 14 --> Insert 8



Delete 16 --> Insert 18 --> Insert 3



Delete 6 --> Delete 14

Question 2

a)

0	26
1	
2	54
3	
4	17
5	69
6	45
7	58
8	32
9	60
10	
11	
12	64

$$\text{Load Factor} = \alpha = \frac{\text{Current number of items}}{\text{Table size}}$$

$$\alpha = \frac{9}{13}$$

For linear probing:
Average number of probes that a search requires:

$$\text{Successful search} \Rightarrow \frac{1}{2}\left[1 + \frac{1}{1-\alpha}\right] = \frac{17}{8} = 2.125$$

$$\text{Unsuccessful Search} \Rightarrow \frac{1}{2}\left[1 + \frac{1}{(1-\alpha)^2}\right] = \frac{185}{32} = 5.78125$$

b)

0	26
1	
2	54
3	
4	17
5	69
6	45
7	58
8	60
9	
10	32
11	
12	64

$$\text{Load Factor} = \alpha = \frac{\text{Current number of items}}{\text{Table size}}$$

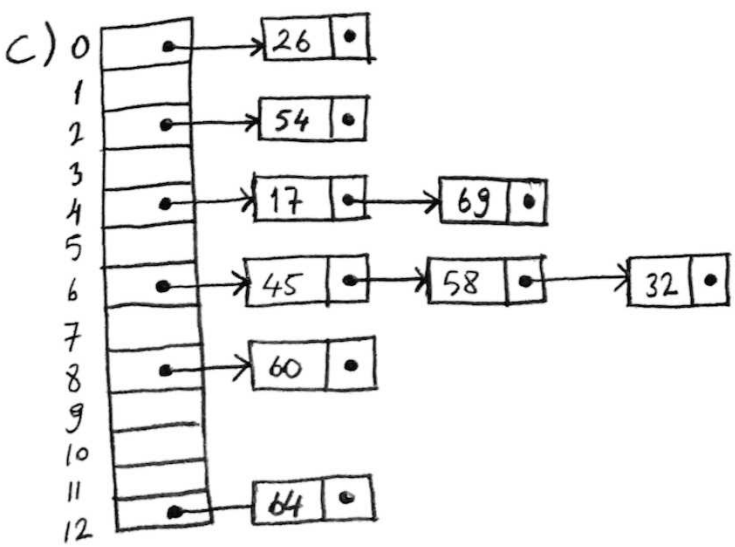
$$\alpha = \frac{9}{13}$$

For quadratic probing:
Average number of probes that a search requires:

$$\text{Successful Search} \Rightarrow \frac{-\log_e(1-\alpha)}{\alpha} \cong 1.7025$$

$$\text{Unsuccessful Search} \Rightarrow \frac{1}{1-\alpha} = 3.25$$

c)



$$\text{Load Factor} = \alpha = \frac{\text{Current no. of items}}{\text{Table size}}$$

$$\alpha = \frac{9}{13}$$

For seperate chaining:
Average number of probes that a search requires:

$$\text{Successful Search} \Rightarrow 1 + \frac{\alpha}{2} \cong 1.3462$$

$$\text{Unsuccessful Search} \Rightarrow \alpha \cong 0.6923$$

Question 3

b)

- **Time Complexity Analysis for the *"Insertion"* Operation:**
  Insertion operation is the process of adding an edge between two vertexes in the graph. The IDs of these two vertexes are taken from the user. Therefore, there is no need to do an extra process to determine the vertexes. Each vertex object keeps its edges in a linked list. That is, when an edge is wanted to be added to these vertexes, the linked list insertion is performed. Usually, this operation can be costly, but each new edge can be added to the linked list's head as a design choice. The worst-case time complexity of inserting to the beginning of a linked list is $O(1)$. Therefore, the worst-case time complexity for the "Insertion" operation in this graph implementation is $O(1)$.

- **Time Complexity Analysis for the *"List"* Operation:**
  List operation lists the edges emerging from a given vertex. In this context, the list operation lists the flights from airport u. It has been talked about before that each vertex stores the edges it has in a linked list. Printing all of the elements in a linked list is bounded by the number of its elements. Hence, the worst-case time complexity for the "List" operation in this graph implementation is $O(|E|)$, where $|E|$ is the total number of possible edges.

- **Time Complexity Analysis for the *"Shortest Path"* Operation:**
  Shortest path operation determines the shortest path and its length between two given vertexes. This operation specifically uses Dijkstra's Shortest Path algorithm. Also, in the implementation of Dijkstra's Shortest Path algorithm, a min-heap is used to select the least costly edges. As is known, Dijkstra's Shortest Path algorithm visits all vertexes in the graph, just like a traversal algorithm. The time complexity of this is $O(|V| + |E|)$, where $|V|$ is the total number of vertices and $|E|$ is the total number of edges. In addition, at each step of the traversal, if a new shorter edge is found, that edge is added to the min-heap. This insertion to heap creates an extra $O(\log|V|)$ time complexity. Therefore, the worst-case time complexity for the "Shortest Path" operation in this graph implementation is $O((|V| + |E|) * \log|V|)$. If the graph is dense, such that $|E| \gg |V|$, the expression becomes $O(|\mathrm{E}| * \log|V|)$.

- **Time Complexity Analysis for the *"Minimize Costs"* Operation:**
  Minimize Costs operation is a process aimed at ensuring access to all airports with minimal costs without losing connectivity. In other words, the minimum spanning tree is determined for the given graph. Prim's Algorithm is also used for this. Similar to the Dijkstra algorithm, in the Prims algorithm every vertex is visited and a min-heap is used to select between shorter edges. The cost of traversal is $O(|V| + |E|)$, and the operations performed on heap (insertion, decrease key, etc.) take $O(\log|V|)$, where $|V|$ is the total number of vertices and $|E|$ is the total number of edges. The difference of Prim's Algorithm is that as the traversal passes through the shortest edges, it saves them in a result array. In asymptotic notation, this operation of writing to the result array is negligible. As a result, the worst-case time complexity for the "Minimize Costs" operation in this graph implementation is $O((|V| + |E|) * \log|V|)$. Again, if the graph is dense, such that $|E| \gg |V|$, the expression becomes $O(|\mathrm{E}| * \log|V|)$.