Problem 1

The given set of keys are: $42, 45, 7, 61, 32, 4, 13, 27, 48$

Linear Probing

For linear probing we take the $key(k)$ and add i to it, then take the 16^{th} modulo of the result. For every key i will be equal to 0. If a key is already present at the position of the hash table then it is considered a collision. For every collision we increase the value of i by 1. The end result is as shown in the figure below. There were a total of *3* collisions for the given st of keys.

Figure 1: Linear Probing

Homework 3

Quadratic Probing

For quadratic probing we take the $key(k)$ and add $\frac{i}{2} + \frac{i^2}{2}$ to it, then take the 16^{th} modulo of the result. For every key i will be equal to 0. If a key is already present at the position of the hash table then it is considered a collision. For every collision we increase the value of i by 1. The end result is as shown in the figure below. There were a total of *4* collisions for the given st of keys.

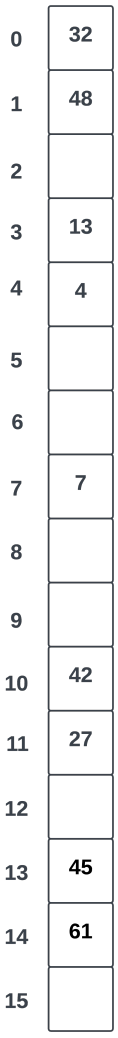

Figure 2: Quadratic Probing

Double Hashing

For double hashing we take the $key(k)$ and add $i \cdot [(k \mod (16-1)) + 1]$ to it, then take the 16^{th} modulo of the result. For every key i will be equal to 0. If a key is already present at the position of the hash table then it is considered a collision. For every collision we increase the value of i by 1. The end result is as shown in the figure below. There were a total of *6* collisions for the given st of keys.

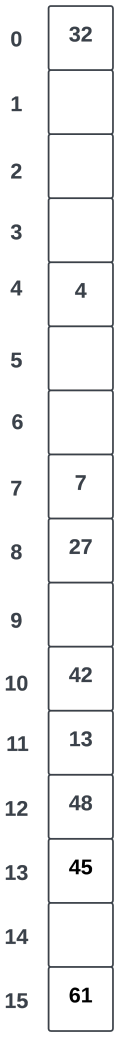

Figure 3: Double Hashing

Homework 3

Problem 2

Algorithm 1: Tree Delete

```
1  TREE-DELETE(T, z)
2      if z.left == NULL
3          TRANSPLANT(T, z, z.right)
4      end
5      else if z.right == NULL
6          TRANSPLANT(T, z, z.left)
7      end
8      else y = TREE-MAXIMUM(z.left)  # Changed
9          if y.p != z
10             TRANSPLANT(T, y, y.right)
11             y.right = z.right
12             y.right.p = y
13         end
14         TRANSPLANT(T, z, y)
15         y.left = z.left
16         y.left.p = y
17     end
```

Algorithm 2: Transplant

```
1  TRANSPLANT(T, u, v)
2      if u.p == NULL
3          T.root == v
4      end
5      else if u == u.p.left
6          u.p.left = v
7      end
8      else
9          u.p.right = v
10     end
11     if v != NULL
12         v.p = u.p
13     end
```

Algorithm 3: Tree Maximum

```
1  TREE-MAXIMUM(x)
2      while x.right != NULL
3          x = x.right
4      end
5      return x
```

Problem 3

To get the maximum number of red nodes we need every other level of the tree to be red. For the case of 9 nodes the tree will look as follows and has 4 red nodes and 5 black nodes.[1]

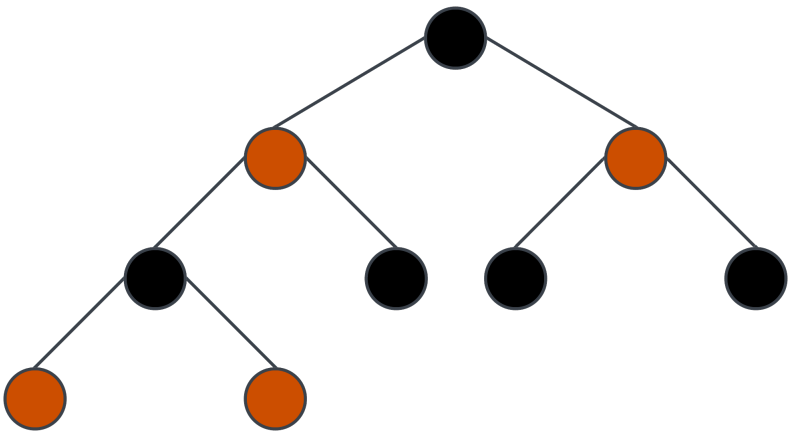

Figure 4: Maximum Red Nodes

To get the maximum number of black nodes we need every node of the tree to be black. For the case of 9 nodes the tree will look as follows and has 0 red nodes and 9 black nodes.[1]

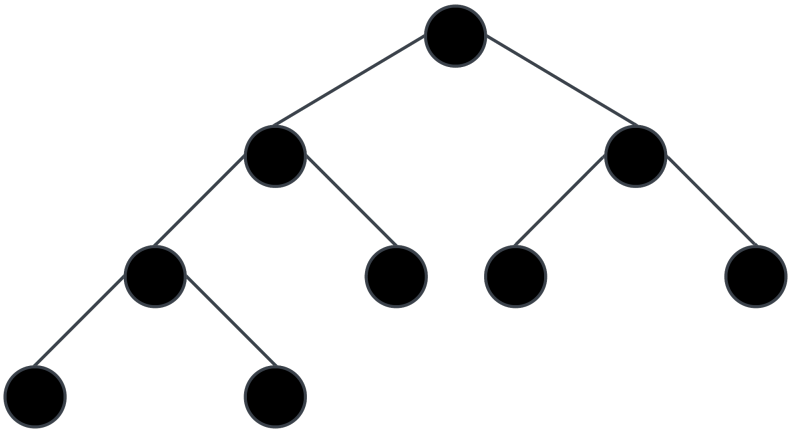

Figure 5: Maximum Black Nodes

Problem 4

4.a: No Preempt

Consider the given case in the question. Increase the number of processes to 3. $P_1 = 5, P_2 = 3, P_3 = 4$. Considering all the processes are ready for execution at $t = 0$. Take P_1 first then P_2 then P_3. The time for each process will be as follows

$$P_1 = 5$$

$$P_2 = 3 + P_1 = 3 + 5 = 8$$

$$P_3 = 4 + P_2 = 4 + 8 = 12$$

Therefore the mean time will be

$$\frac{P_1 + P_2 + P_3}{3} = \frac{5 + 8 + 12}{3} = 8.333$$

But if they are processed in a sorted order then we get the times as follows

$$P_2 = 3$$

$$P_1 = 4 + P_2 = 4 + 3 = 7$$

$$P_3 = 5 + P_1 = 5 + 7 = 12$$

Therefore the mean time will be

$$\frac{P_1 + P_2 + P_3}{3} = \frac{3 + 7 + 12}{3} = 7.333$$

As we can see sorting the processes according to their process time reduces the mean time required for each process.

From this we can deduce that the time complexity to process all the processes will be the same as the sorting algorithm used. The best average time complexity can be achieved from algorithms like quick sort and merge sort which will be $O(nlogn)$

4.b: With Preempt

If we sort all the processes according to their processing time just like in 4.a We might get a case where in the next task cannot be processed as it's release time is in the future. Hence we must find a task which can be processed by iterating over the list. This iteration will have a worst case time complexity of $O(n)$. Since this will be done every time a process is preempted and a new task is to be processed the total time complexity of will become $O(n^2)$

References

1. Jindal, Pawan, Amit Kumar, and Shishir Kumar. "Minimum and maximum ratio of number of red internal nodes to black internal nodes in Red Black tree." International Journal of Computer Applications 975 (2010): 8887.