Sam Kelly - 173943

ASD Exam – Sam Kelly 18th May 2022

Question 1. A)

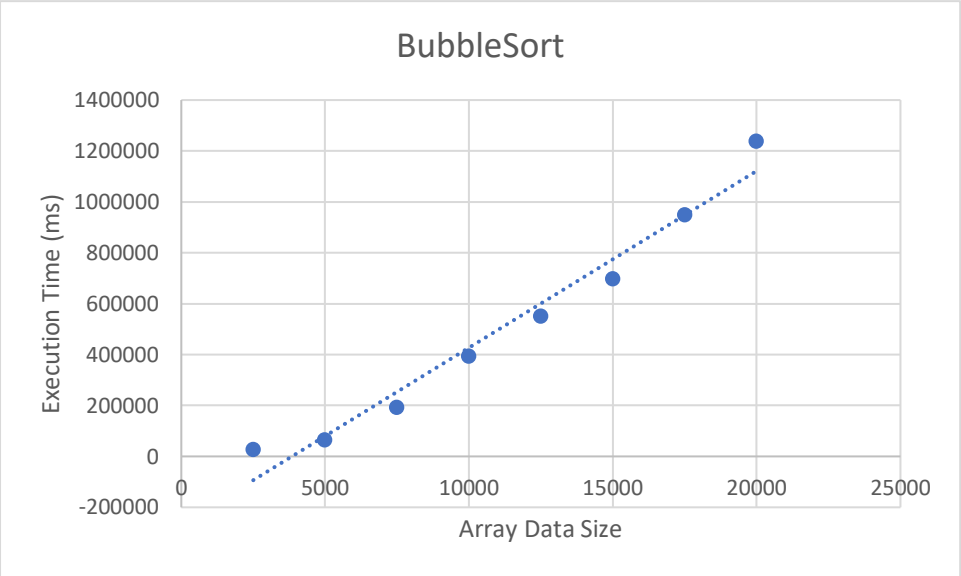


Figure 1. Graph showing the BubbleSort execution times.

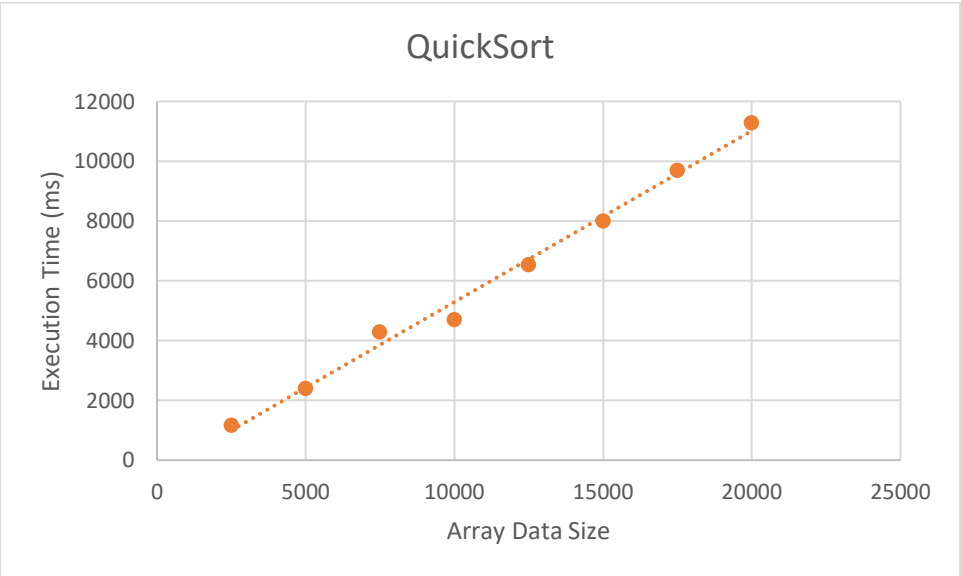


Figure 2. Graph showing the QuickSort execution times.

In both charts we can see that as the size of the data increases so does the time taken in milliseconds for the functions to execute. However, We can see that BubbleSort is significantly slower when the array data size gets larger. As we can see from the BubbleSort graph the execution time increase exponentially this is because the Big O of bubble sort is O(n^2) which means that if the value of n is 2 the loop will run 4 times. It is this property of bubble sort that makes the execution time so much slower than quick sort with larger data

sets. The QuickSort function has a Big O of O(n log n) which leads to much faster results than the bubble sort.

Question 1. B)

Bubble sort has a time complexity of O(n^2). This is because bubble sort has an inner and outer loop, the inner loop does O(n) on each of the iterations and the outer loop runs for O(n) iterations.

Quick sort in this case has a time complexity of O(n^2). This is because there is a pivot at the start of the array.

In the final case Quick sort has a time complexity of O(n log(n)). because the partition is set in the middle.

Question 1. C)

i. If a program allows you to call a function inside of a function then it is called a recursive function. In the example of this quicksort the function of setting the pivot and then moving the data is done inside of the function to check if all the data has been sorted correctly.

ii. In this quicksort example recursion occurs in the second while loop “WHILE left <= right AND array[left] <= pivot SET left TO left + 1 END WHILE”. Here the recursion is occurring while the data in the left is less than or equal to the right and the left of the array is less than the pivot set.

Question 2 A).

A depth first search on the graph would be done in the order of A -> **B** -> D -> E -> H -> L -> M -> N -> I -> O ->P -> C -> F -> G -> J -> K -> Q. The assumptions made in order to get this order nodes is that the start point is A and we are adding from right to left.

The depth first search algorithm is used for searching a tree data structure. It will start at the root node and will explore as far as it can along a branch before backtracking and starting another branch. In theory depth first search will take O(V+E) time to traverse a graph where V is the number of vertices in the graph and E is the number of edges. In some cases the depth first search may use O(V) to store the vertices along the current search path and set them as visited. This means that when O(V) is used the depth first search is similar to a breadth first search. In some applications a depth first search will run into a problem called “non-termination” this means that the graph being traversed is so large it will never reach the end. In these cases a limited depth is applied to the algorithm to stop it from reaching this state. This is one of the major limitations of the depth first search algorithm. The depth first search can also use a heuristic method to choose where to go next.

Sam Kelly - 173943

Question 2 C).

A queue is a first in first out data structure. The two main operations used on a queue are add() or enqueue() and remove() or dequeue(). Firstly add or enqueue, this operation will add an item the queue it does this by adding the data and then storing it in the queue. The second operation used on a queue would be the remove or dequeue operation. This operation will remove the access to an item from the queue essentially removing or kicking the item out.

Question 3 A).

Pre-order traversal: First: 77 -> 40 -> 39 -> 42 -> 86 -> 85 -> 90 Last

Post-order traversal: First: 39 -> 42 -> 40 -> 85 -> 90 ->86 -> 77 Last

In-order traversal: First: 39 -> 40 -> 42 -> 77 -> 85 -> 86 -> 90 Last

Question 3. B)

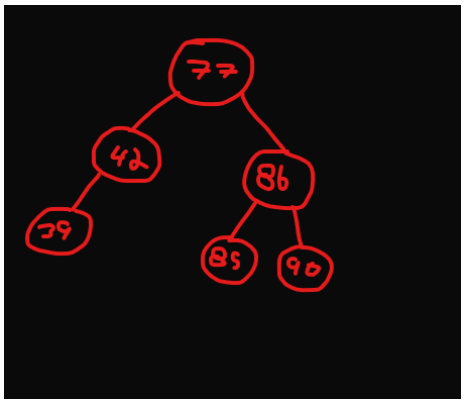
When inserting data into a tree you first need to locate where it would go by searching from the root. If the data is less than the root node we search the left values. Because 65 is larger than the left node and smaller than the right node it will be inserted above the left node. In total the traversal of the 65 node would start alongside the 40 and then the 40 node will be pushed down the tree because it's value is lower than that of the new 65 node leaving the 65 node as the new node in the left tree.

If we were to search for the node 39 we know 39 is less than 77 so we can start on the left branch and then continue this down the tree. 39 is less than 65 so we keep going, 39 is less than 40 so we would go down the left branch and arrive at 39. This would give us O(log n) in the best scenario. However, considering the space complexity of a binary tree, the number of nodes in the tree increases the amount of space taken up in memory making it harder and harder to search. Therefore in the worst case scenario the Big O of this tree would be O(n).

Sam Kelly - 173943

Question 3. C)

To delete the number 40 we would need to find the successor of the node. This is done in order so if we were to delete 40 from the tree the next highest value would take its place in this case 42 so when re drawing the tree with the deleted 40 node it would look like this;



Question 4. A)

Hashing is a way to store data in a table known as a hash table. To store data in this table it has to have a numerical value because of this non numerical data is given it's ASCII code. The ASCII code is then used and sent through a hash function in order to generate an appropriate index number, which is then used to store the data in the hash table. This process is known as hashing.

Question 4. B)

36 mod 9 = 0

(0+0) mod 9 = 0

Therefore 36 mod 9 = hash[0] so 36 would be inserted into the table as follows;

0	36
1	
2	
3	8
4	23
5	
6	
7	
8	
9	
10	

Sam Kelly - 173943

Question 4. C)

36 mod 9 = 0

(0+0^2) mod 9 = 0

Therefore 36 mod 9 = hash[0] so 36 would be inserted into the table as follows;

0	36
1	
2	
3	8
4	23
5	
6	
7	
8	
9	
10	

This approach does solve primary clustering however, it does not solve secondary clustering this is because two keys can have the same probe sequence and still hash to the same location. Some level of clustering is unavoidable due to the nature of having limited spaces in the table. Primary clustering happens in linear probing where the collision resolution creates slots close to the hash position of keys. Secondary clustering has the same effect but creates these runs of filled slots away from the hash position of a key. Therefore while quadratic probing does solve primary clustering it creates secondary clustering.

Question 4. D)

Separate chaining is when you would make each cell in a hash table point to a linked list of other records with the same hash value. This has some advantages mainly it is easy to implement. It is also good because the hash table technically can never fill up as all we do is add more elements to the linked lists. Like all things though separate chaining has some disadvantages. The performance of the cache with this method is not as good as all the keys are stored using linked lists. You will tend to waste a lot of space as not all parts of the hash table are guaranteed to be used and similarly you will have to use extra space to store these links. Finally when your chain of links because long you will have severely slowed search times and in the worst case can become O(n).

END OF PAPER