Berat Efe Ceylan

123200073

CMPE211

Part 1: Sorting Algorithms

Instance 1: Merge Sort performs very well due to its stability and consistent performence it always works in O(nlogn) regardless of input. But on the other hand QuickSort can reduce its performance because of the input. Average complexity is same as merge sort O(nlogn) but if array is already sorted it will be O(n^2) if its nearly sorted again it wont be directly nlogn. This happens because of pivot choice consistently results in unbalanced partitions. We can prevent it by randomized pivot selection.

- **Size:** 9
- **Nature:** Nearly sorted list.
- **Input:** {2,3,7,9,10,15,18,13,20}.

Instance 2: Merge Sort performs good but it is slightly slower because of overheads in merging. Quick Sort performs better for random inputs due to efficient partitioning.

- **Size:** 9
- **Nature:** Random.
- **Input:** {4,3,7,0,1,14,12,2,6}.

Sorting Algorithms			
Sort Type	Instance 1 times	Instance 2 times	Average Times
Insertion Sort	718300 ns	8500 ns	363400 ns
Quick Sort	1239100 ns	6100 ns	622600 ns

Part 2: Search Algorithms

Sequence 1:

Since there is a collision due to 2 Put 5 the time complexity of HASH will be O(n) from O(1) but in BST its always O(logn) so its faster than HASH here.

Put 5

Put 10

Put 5

Get 10

Get 5

Sequence 2:

Generally hash table performs well even with interleaved operations,operations depend on efficient hash computation not structural traversal. Non exist keys like 3 require minimum probing so it is much faster then BST which do full branch traversal. In hash get and put O(1) but for BST its O(logn) if its balanced,if not O(n)

Put 2

Put 4

Put 1

Get 3

Get 4

Put 6

Get 6

Searching Algorithms			
Search Type	Sequence 1 times	Sequence 2 times	Average Times
Binary Search Tree	609600 ns	119600 ns	364600 ns
Hash Table	859800 ns	87500 ns	472750 ns