2008 Paper 11 Question 7

1. The partition procedure, takes the list and splits it into lists. One list contains elements smaller than the ‘pivot’ the other larger. This pivot is an element of the list which has been picked (could be random for better results), which the separates the two produced lists. i.e. ‘smaller list’ ‘pivot’ ‘larger list’. The pivot is normally the last element of the list

Quicksort(List L1)

If (L1.length == 1 )then return L1

Else …

Partition(L1) -> Lists L2,L3 + Element Pivot

QuickSort(L2)

QuickSort(L3)

return (L2 + Pivot + L3) + operator adds it all into one list

L2 is smaller than pivot and L3 is larger than pivot

1. Worst Case – That the pivot is always the smallest or largest element (ie list is sorted already)

Therefore Partition will create from a list of n elements a list of 1 element and a list of n-1 elements and then that into a list of 1 and n-2 etc.

Each time the program partitions it takes kn time where n is length of list. Therefore time complexity = k( n + n-1 + n-2 +…+1), which is O(n^2) time complexity.

To resolve this, the pivot should be taken at random or be a median of a set of numbers from the list, therefore guaranteeing that the pivot isn’t the smallest or largest number in the list.

1. Taking a random pivot from array

Adv- If the list is already sorted, it will prevent the worst case of O(n^2) from happening since the chances that the pivot is the largest/smallest each time is miniscule. It works by pre-shuffling the list so that the algorithm can work better as most lists are normally semi sorted

Dis- It adds an extra computation to algorithm and for some languages, getting a random variable is a costly computation.

For worst-case it makes no difference, but it reduces the chances of getting worst case. For Average case there is an increase but the difference is negligible as getting a random element is normally O(1)

1. The median of an array of n numbers is the middle element of a sorted list. i.e. n/2th element of the list.

Median(List L1, Middle)

If (L1.length == 1 )then return L1[0]

Else…

Partition(L1) -> Lists L2,L3 + Element Pivot

If (Middle – L2.length == 1) return Pivot

Else

If (Middle > L2.length)

return Median(L2, Middle)

Else

return Median(L3, Middle – L2.length - 1)

L2 contains elements smaller than pivot, L3 larger

Idea is that Partition the list and the Median will either be in one list or the other or it will be the pivot. Then recursively call Median in the List which contains the Median.

Best Case is O(1) ie the pivot of first call is the Median

Worst Case

F(n) = f(n/2) + kn

= f(n/4) + k(n + n/2)

= f(n/8) + k(n + n/2 + n/4) etc

= k(n + n/2 + n/4 + n/8 + …) = 2kn

Therefore Worst Case time Complexity is O(n)