DATA 311 ASSIGNMENT 2

Yunfan Yang 30067857

Quick Sort

Description

Quicksort is a divide-and-conquer sorting algorithm, and it can be implemented by using recursion. A partition is a divided sub-array, and a pivot is a flag point chosen by the algorithm in a partition for conquest. Quicksort executes the following steps:

1. Pick an element from the given array as a pivot. This pivot element can be random or designated.
2. Partitioning the given array: starting from both side, conquer the two value on the left and the right side with the pivot value; if the left value is smaller than the pivot, or if the right value is greater than the pivot, keep moving to the next element on the side; until one element from the left side is greater than the pivot, meanwhile one element from the right side is smaller than the pivot, then exchange them and keep moving to the next element on both sides. The element equals to the pivot can go either side. Eventually, both left and right conquest index will meet each other at one element, and as a result, all the elements on the left of the pivot point will be smaller than the pivot and all the elements on the right of the pivot point will be greater than the pivot point.
3. Divide the given array into two sub-arrays, and the element which left and right searching point meet is the point of division. For each of the divided sub-arrays, recursively apply the above steps.

The base case of Quicksort is the left conquest index is greater than or equal to the right conquest index, which means all the elements in the given array are partitioned.

The best, average and worst case are: $O(n \log n)$, $O(n \log n)$ and $O(n^2)$. The selection of the pivot element can affect the Big-O since it might be the smallest or greatest element which leads to one sub-array only has one element and increase the times of recursion.

Asymptotic Complexity

$$T(n) = 2T(\frac{n}{2}) + \theta(n)$$

$$a = 2, b = 2, f(n) = \theta(n)$$

$$log_b a = log_2 2 = 1 = \theta(n)$$

$$\theta\left(n^{log_b a} \log n\right) = \theta(n \log n)$$

Merge Sort

Description

Merge sort is a divide-and-conquer sorting algorithm as well and it can be implemented by using recursion. This algorithm divides the given array into all sub-arrays, then sort by conquest and merge the sub-arrays. Merge sort executes the following steps:

1. Divide the given array into two sub-arrays by half, recursively apply this step until there is only one element to divide. The divided will apply the next step.
2. Sort and merge the two halves: starting from the first element of both halves, conquer both elements from both sides, the greater element will be copied to the new array first, and move to the element of the side, then continuing conquest; until there is no element left for conquest on one side, copy the rest of elements on the another side to the new array.
3. After merging the two sub-arrays, recursively continuing sort and merge with the upper-level sub-arrays, until there is only one whole array.

The base case of Merge sort is the left conquest index is greater than the right conquest index, which means the given array cannot be divided anymore.

The best, average and worst for Merge sort are all $O(n \log n)$.

Asymptotic Complexity

$$T(n) = 2T(\frac{n}{2}) + \theta(n)$$

$$a = 2, b = 2, f(n) = \theta(n)$$

$$log_b a = log_2 2 = 1 = \theta(n)$$

$$\theta\left(n^{log_b a} \log n\right) = \theta(n \log n)$$

Binary Search

Description

Binary search is a searching algorithm which finds the position of a target value in an array or detects whether the given array contains the target value. For using this searching algorithm, the given array must be sorted first. This algorithm can be implemented using recursion. Binary search executes the following steps:

1. Pick a middle point index from the given search range in the given array.

2. If the value to this middle point index is exactly the target value, return the middle point index since the position of the target value is found; otherwise, compare the target value with the value to the middle point index: if the target value is greater, the search range zoom in to the middle point element and the last element of the given range; else, if the target value is smaller, the search range zoom in to the first element of the given range to the middle point element.
3. Recursively apply the above steps until the value is found or the range cannot be zoomed any more.

The base case of Binary search is the left range index is greater than the right range index which means the range is not valid and the target value is not in the given array.

The best, average and worst case for Binary search are: $O(1)$, $O(\log n)$ and $O(\log n)$. The best case means that the first middle point value exactly is the target value.

Asymptotic Complexity

$$T(n) = T\left(\frac{n}{2}\right) + \theta(1)$$

$$a = 1, b = 2, f(n) = \theta(n)$$

$$log_b a = log_2 1 = 0 < \theta(1)$$

$$\theta\left(n^{log_b a} \log n\right) = \theta(n^0 log n) = \theta(\log n)$$