Assignment 1

**Question 1:**

A] i)Naïve way:

*public static double naive(double base,int exponent) {*

*double result = 1;*

*for (int i = 0; i < exponent; i++) {*

*result = result \* base;*

*}*

*return result;*

*}*

ii)Divide and conquer:

*public static double divideAndConquer(double base, int exponent) {*

*if (exponent == 0) {*

*return 1;*

*}*

*else if (exponent % 2 == 0) {*

*double halfPower = divideAndConquer(base, exponent / 2);*

*return halfPower \* halfPower;*

*}*

*else {*

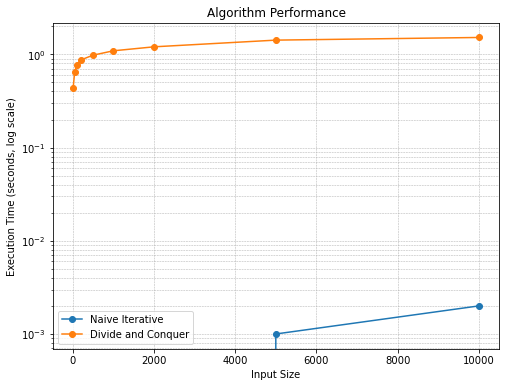
*double halfPower = divideAndConquer(base, (exponent-1) / 2);*

*return base\* halfPower \* halfPower;*

*}*

*}*

B] In the naïve way, the time complexity is O(n) where n is the exponent. In divide and conquer, *T*(*n*)=*T*(​)+*O*(1) , In this tree:The root node represents the original problem . Each internal node has two children, representing the recursive subproblems . The leaf nodes represent the base case of the recursion (*n*=1), where is calculated directly. The height of the tree is .



**Question 2:**

A]

def mergeSort(arr):

if len(arr) <= 1:

return arr

mid = len(arr) // 2

left\_half = mergeSort(arr[:mid])

right\_half = mergeSort(arr[mid:])

return merge(left\_half, right\_half)

def merge(left, right):

i = j = 0

merged = []

while i < len(left) and j < len(right):

if left[i] < right[j]:

merged.append(left[i])

i += 1

else:

merged.append(right[j])

j += 1

merged.extend(left[i:])

merged.extend(right[j:])

return merged

def binarySearch(a, key, i, j):

if j >= i:

mid = (i + j) // 2

if a[mid] == key:

return mid

if a[mid] > key:

return binarySearch(a, key, i, mid - 1)

else:

return binarySearch(a, key, mid + 1, j)

return -1

def findPairs(arr, sum):

sorted\_arr = mergeSort(arr)

pairs = []

for i in range(len(sorted\_arr)):

complement = sum - sorted\_arr[i]

index = binarySearch(sorted\_arr, complement, i + 1, len(sorted\_arr) - 1)

if index != -1:

pairs.append((sorted\_arr[i], sorted\_arr[index]))

return pairs

S = [10, 3, 5, 2, 7, 1, 9, 4, 6, 8]

sum = 10

pairs = findPairs(S, sum)

print("Pairs: " + pairs)

B]

In the merge sort, the array is divided into smaller sublists until each sublist contains only one element which takes O(log(n)). Then, the two sorted lists are merged together; the algorithm compares the elements from two sublists and places them in sorted order. Merging two sublists of size n/2 takes time of O(n). The total time complexity can be expressed as O(n log (n)) because at each level of recursion, it performs linear time merging.

The time complexity of Binary Search in a sorted array is O(log (n)) in the worst case.

The findPairs method performs merge sort first which takes O(n log(n)).Then, for each element in the sorted array, binary search is performed, which takes O(log (n)) time.The total time complexity of the algorithm is O(n log (n) + n log (n)) = O(n log (n)).

The recurrence relation is T(N)=2T(N/2)+O(N)

Level 1: n

Level 2 : n/2 n/2

Level 3 : n/4 n/4 n/4 n4

………………………………..

Level i : 1 1 1 1 ……..

The total number of levels in the tree is log(n) because the problem size is halved at each level.

At each level *i*, there are subproblems

The cose of each level i is *O*(*N*).

Total cost =

By solving the summation, total cost = O(n log(n))

