**Question 1:**

def find\_max\_points(points, n, m):

İ wrote this function for outer function.

def dfs(i, j, current\_points):

This is inner function for recursive. These function iterate one by one i+1,j or i,j+1 situations and sum elements. When finish iteration for one loop, again function sum elements and compare

Previous result of summing elements.

d = {'max\_points': 0}  
f = {'path': []}

Those are my outer function variables for use in inner function. After all process i return those two variable to main and print on console.

**Question 2:**

I use decrease and conquer algorithm with merge sort. I split array with two and merge them. After this process i look array’s length. If length is odd return mid element of array, else i found length/ – 1. and length/2. Element of array. Then i calculate arithmetic mean and return this value. This is my median.

Best case:

One element array is best case for this algorithm. O(1) is time complexity

Worst case:

Worst case of this algorithm’s time complexity is O(n logn).

For testing i create a function for create randomly array. This function first decide size of array randomly and fill this array with numbers which generated randomly.

**Question 4:**

The time complexity of ternary search is 𝑂(𝑙𝑜𝑔3𝑛), which means that the number of steps required to perform the search grows at most logarithmically with the size of the input, but with a lower rate of growth than binary search, which has a time complexity of 𝑂(𝑙𝑜𝑔2𝑛). This means that for large inputs, ternary search may be faster than binary search.

The divisor (in this case, 2 for binary search and 3 for ternary search) affects the time complexity of the search algorithm by determining the rate at which the number of steps required to perform the search grows as the size of the input increases. In general, a larger divisor will result in a slower rate of growth, which means that the time complexity of the algorithm will be lower.

If we divide the array into n parts at the beginning, the time complexity of the algorithm becomes 𝑂(𝑛). This means that the number of steps required to perform the search grows linearly with the size of the input. This is because in this case, the algorithm has to check every element in the array before it can determine whether the target element is present or not. This makes the algorithm very slow for large inputs.

**Question 5:**

**a)**

The best-case scenario for interpolation search is when the target element is present at the first position in the array. In this case, the algorithm will find the target element in a single step, making it the most efficient case.

The time complexity of interpolation search is 𝑂(𝑙𝑜𝑔𝑛) in the average case and 𝑂(1) in the best case. This means that the number of steps required to perform the search grows logarithmically with the size of the input on average, but in the best case, it remains constant.

Interpolation search works by using the values of the elements in the array to estimate the position of the target element. It does this by calculating the position of the target element based on a formula that takes into account the values of the elements at the start and end of the current search range. If the target element is present in the array, interpolation search will typically find it faster than other search algorithms such as linear search or binary search. However, if the array is not sorted or if the values of the elements are not uniformly distributed, interpolation search may not perform as well.

**b)**

Interpolation search and binary search are both search algorithms that are used to find a target element in a sorted array. However, they differ in the way they search for the target element and in their time complexity.

One of the main differences between interpolation search and binary search is the way they choose the next position to search. Binary search works by dividing the array into two halves at each step and choosing the half that is more likely to contain the target element. On the other hand, interpolation search uses a formula to estimate the position of the target element based on the values of the elements at the start and end of the current search range.

Another difference between the two algorithms is their time complexity. The time complexity of binary search is 𝑂(𝑙𝑜𝑔2𝑛), which means that the number of steps required to perform the search grows logarithmically with the size of the input. On the other hand, the time complexity of interpolation search is 𝑂(𝑙𝑜𝑔𝑛) in the average case and 𝑂(1) in the best case. This means that the number of steps required to perform the search grows logarithmically with the size of the input on average, but in the best case, it remains constant.

Overall, interpolation search may be faster than binary search for large inputs, especially if the values of the elements in the array are uniformly distributed. However, binary search may be more reliable in cases where the array is not uniformly distributed or if the values of the elements are not known beforehand.