CSE321 HW4

Q1

The algorithm recursively passes through both east and south neighborhoods. Then on the way back it sums up the values of the neighborhoods that were passed through in each recursive call sum of both the east and south ways are compared and the greater one is returned. If the question was to only to find the greatest point that is achievable under these constraints only returning the greater sum of east and south ways would be enough, but to print the path that was taken and the distinct points of cells the algorithm written for this problem returns those, but the core idea is same.

The worst-case scenario for this algorithm is same as its best case since we need both the east and the south sums to compare them making it O(nm).

Q2

The algorithm to find median is simply a modified quick selection algorithm. The only differences are the new base conditions regarding the evenness of the number of elements in given array, since median of even number of integers is defined as average of middle two elements.

The worst-case time complexity for this algorithm is the same as quick selection's. Therefore, it is O(n^2). Since we know it is equivalent to the quick selection algorithm in terms of time complexity, we can also say that its average time complexity is O(n).

Q3

a)

The circular linked list data structure is implemented using Python's list data structure since it is convenient. Assuming addition and removal operations of Python's list is constant time the algorithm has a linear time complexity. The algorithm deciding the winner basically gets to the next player and removes them from the list. This is done until there is only one player is left.

So, each whole loop of the list half of it gets removed, so in each loop there's only half of iterations compared to the previous loop. Therefore, its time complexity is

$$\frac{n}{2}+\frac{n}{4}+\frac{n}{8}+\cdots=n \text{ As n goes to inf i nity}$$

b)

To find an algorithm to solve this problem in logarithmic time I first observed the numbers of winners for each given number of players for the numbers from 1 to 30. Then I noticed a pattern and worked out the exact formula for given number of players:

$$T(n)=2\cdot\left(n-2^{\lfloor \log_2 \quad n\rfloor}\right)$$

Starting from this formula I tried to break it down into a recursive formula, after a couple of trial and errors I arrived to

$$T(n) = 2 \cdot \big(n\%2 \ + \ 2 \ \cdot \ T(n-1)\big)$$

$$T(1) = 0$$

Q4

Even though the time complexity of ternary search is $O(\log_3 \quad n)$ it is worse due to its constant factor. For every recursive call or iteration, it needs to make two comparisons since both are logarithmically complex the change rate is equivalent but due to ternary search's greater number of comparisons it is slower.

Dividing the array into more parts increases the base of the logarithm (making it faster) but also the number of comparisons in each step (making it faster). When dividing the array is pushed to the extreme (n-division) the overall time complexity would be linear, checking every element in order.

Q5

a)

The best-case scenario would be having a list of items such that the difference in any consecutive pair is always same. Such as [1, 2, 3, 4, 5] or [5, 12, 19, 26]. In this scenario, the algorithm would find the exact position of the target in the first step. So best case time complexity is constant time.

b)

The interpolation search relies on the structure of the data, it targets to be effective where the data is linearly changing. On the other hand, binary search does not rely on any structure of the data. So, if an array is suitable for interpolation search, then it will be searched more quickly by interpolation search. But whenever this is not possible the binary search is going to be faster. The best case of interpolation search is better than binary search's but the worst cases are the opposite.