Problem 1.

The algorithm is a modified version of merge sort with extra condition checking process which takes O(1) during merge stage. The algorithm is as following

```
Problem1(A, K, L, results=[]):
    if len(A) ≤ 1 :
        return A, results

    # split A into 2 about equal halves
    mid = len(A) // 2
    left_half = A[:mid]
    right_half = A[mid:]

    # recursively sort both halves and collect pairs that satisfy
      the conditions asked by the problem
    left_half, L_r = Problem1(left_half, K, L, results)
    right_half, r_r = Problem1(right_half, K, L, results)

    # updates results from returned vals.
    results += L_r + r_r

    return merge(left_half, right_half, K, L, results)
```

merge see next page

```
merge(left, right, K, L, results):
    merged = []
    left_idx, right_idx = 0, 0

    while left_idx < len(left) and right_idx < len(right):
        if left[left_idx] < right[right_idx]:
            merged.append(left[left_idx])
            if satisfy(left[left_idx], right[right_idx], K, L):
                append the indices of the pair to results
            left_idx += 1
        else:
            merged.append(right[right_idx])
            if satisfy(right[right_idx], left[left_idx], K, L)
                append the indices of the pair to results
            right_idx += 1

    # append all remaining elements from both halves
    merge.extend(left[left_idx:])
    merge.extend(right[right_idx:])

    return merged, results
```

```
satisfy(smaller_num, bigger_num, K, L):
    M = (smaller_num + L - 1) // L * L   ← smaller number divisible by L
    upper_bd = M + K * L - 1                that's greater than smaller_num
    if smaller_num + 1 ≤ bigger_num ≤ upper_bd:
        return True
    return False
```

Time complexity:

Since it uses merge sort with two additional condition check of time $O(1)$, the time complexity is $O(n \log n)$.

Space complexity:

The space complexity of merge sort is $O(n)$ and the extra array 'results' is of length n. Then, total space complexity is $O(n) + O(n) = O(2n) = O(n)$.

Problem 2.

```
Search (A, x, i=1, j=m):
    if i<1 or i > n or j <1 or j>m :  # out of bound
        return False
    if A[i,j] == x , then return True .
    if x>A[i,j], search (A, x, i+1, j)
    elif x < A[i,j] , search (A, x, i, j-1)
```

Time complexity:

Worst case is when the algorithm iterate all last column and all last raws, which results in O(m+n)

Space complexity:

Space complexity is O(1) since there is no extra space used.

Problem 3

Similar to the algorithm given in problem 1. This is a modified version of merge sort. During the merge stage, only

```
if left_half[left_idx] > right_half[right_idx]
    count += 1
```

Time complexity:

Since it uses merge sort with two additional condition check of time O(1), the time complexity is $O(n \log n)$.

Space complexity:

The space complexity of merge sort is $O(n)$. There is an additional variable stores counts. Then, total space complexity is $O(n) + O(1) = O(n)$.

Problem 4

1. Upper_bound is $\frac{n}{2}$ when for all individuals, there is exactly 1 empty seat between any pair of individuals. □x□x□

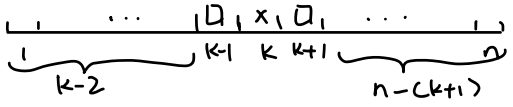
   Lower_bound is $\frac{n}{3} + n\%3$ when there are exactly 2 empty seats between any 2 individuals □x□□x□

There are at least 1 empty seat and at most 2 empty seats between 2 people according to the statement of the problem. In order to have the maximum number of individuals sit in n seats, there should be exactly 1 seat between each pair. Similarly, to have the minimum number of individuals, there should be exactly 2 seats between each pair. (and we can think (empty, occupied, empty) as a group of 3. that's disjoint from other groups)

2. For n available seat,

$$T(n) = \frac{1}{n}\sum_{k=1}^{n}(1 + T(k-2) + T(n-(k+1)))$$

where for $n \leq 0$, $T(n) = 0$



Supposed the position k is taken where k can be anywhere from 1 to n. Then, there is $\frac{1}{n}$ chance that Kth position is picked. Then, K-1 and K+1 should be empty according to COVID policy. So, we only needs to consider the first K-2 and the last $(n-(k+1)) = n-k-1$ seats, which will be taken care of by recursion. Eventually, base cases will be approached.

Problem 5

$\mathbb{E}(T) \leq T(n)$ ← run time

$\Pr(\text{correct}) = \gamma(n)$

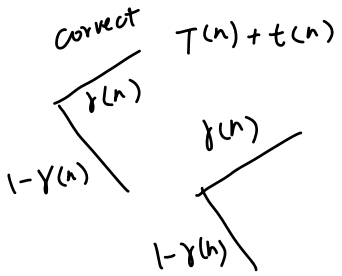
time to verify correctness $= t(n)$

What is expected run time of obtaining a Las Vagas algor that always gives correct answer

1. run the algorithm $\mathbb{I}$. ← $T(n)$
2. verify correctness ← $t(n)$

$\gamma(n)$ ↙ return

$1-\gamma(n)$ ↘ step 1.



Let X be the number of times of step 1 and 2 need to be done. where

$X \sim \text{Geometric}(\gamma(n))$

Then, $\mathbb{E}(X) = \frac{1}{\gamma(n)}$

So, the expected runtime is $\mathbb{E}(X) \times (T(n)+t(n)) = \frac{1}{\gamma(n)}(T(n)+t(n))$.

Problem 6.

$Pr(H) = p \quad Pr(T) = 1-p$

If toss the coin twice, $Pr(HH) = p^2$. $Pr(TH) = Pr(HT) = p(1-p)$, $Pr(TT) = (1-p)^2$

Then, if get both heads or both tails, retoss.

Else, if get TH then it is HEAD. If get HT, then it's TAIL.

$$Pr(\text{HEAD}) = Pr(\text{TH} \mid \text{2 tosses have different results}) = \frac{p(1-p)}{2p(1-p)} = \frac{1}{2}$$

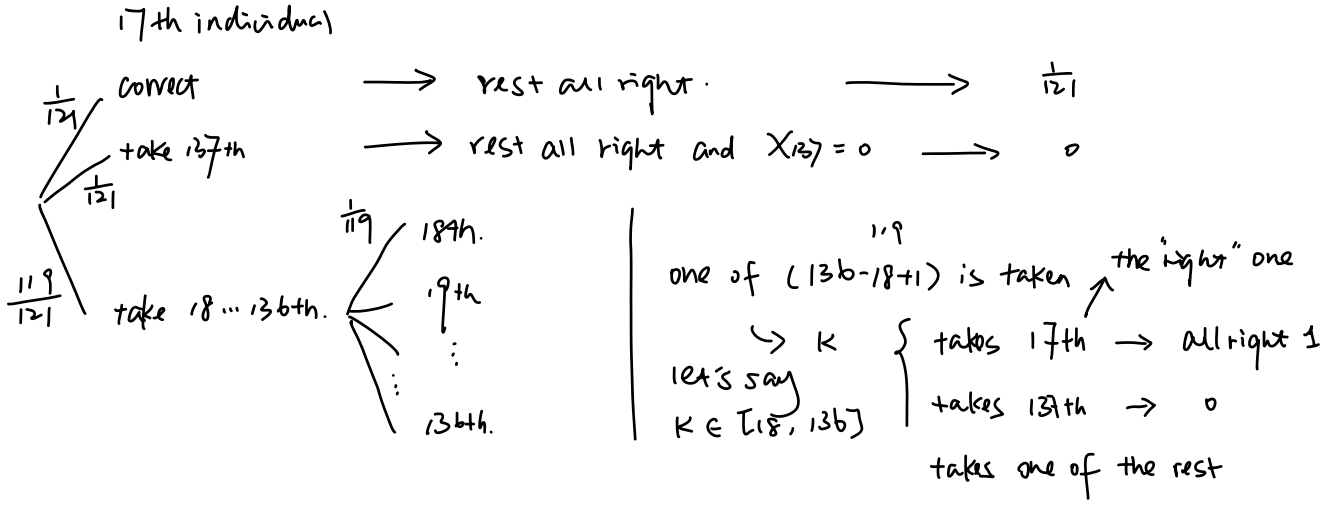
$$Pr(\text{TAIL}) = Pr(\text{HT} \mid \text{2 tosses have different results}) = \frac{p(1-p)}{2p(1-p)} = \frac{1}{2}$$

Let X be the number of pairs of tosses where $X \sim \text{Geometric}(2p(1-p))$

Then, $E(2X) = \frac{1}{2p(1-p)}$

Thus, the expected number of tosses is $2 \times E(2X) = \frac{2}{2p(1-p)} = \frac{1}{p(1-p)}$

Problem 7.



So, the senerios can be generalized as below:

for ith individual where $i \in [17, 136]$

Pr (i takes 137th seat) $= \frac{1}{137-i+1} = \frac{1}{138-i}$ → 137th can't take his seat

Pr (i takes the right seat) $= \frac{1}{138-i}$ → 137th can for sure get correct seat.

* by "right", it means i takes someone else who took his seat or the correct seat for i = 17.

Pr (i takes kth seat where $k \in [i+1, 136]$) $= 1 - \frac{2}{138-i}$

Let the probability that the kth individual leads the 137th individual to sit on his assigned seat is P. Then,

$$P = \frac{1}{121} \times 1 + \frac{1}{121} \times 0 + \frac{121-2}{121} \times P$$

$$P = \frac{1}{121} + \frac{119}{121} \times P$$

$$\frac{2}{121} P = \frac{1}{121}$$

$$P = \frac{1}{2}$$

Problem 8: Randomized-QS expected runtime of each recursive call.

$$T(n) = n - 1 + \frac{1}{n}\sum_{rank=1}^{n} T(rank-1) + T(n-rank)$$

1. Since in RANDOMIZED-QUICKSORT, each pivot is selected at random and there are n elements, the probability that any particular element is chosen as the pivot is $\frac{1}{n}$.

2. $T(n) = n-1 + \frac{1}{n}\sum_{i=}^{n-1}(T(i)+T(n-i))$ ↘ by symmetry

$= n-1 + \frac{2}{n}\sum_{i=0}^{n-1} T(i)$

↓

$E(T(n)) = E(n-1 + \frac{2}{n}\sum_{i=0}^{n-1} T(i))$

$E(T(n)) = n-1 + \frac{2}{n}\sum_{i=0}^{n-1} E(T(i))$

$E(T(n)) = \frac{2}{n}\sum_{q=2}^{n-1} E(T(q)) + \frac{2}{n} * E(\cancel{T(0)} + \cancel{T(1)}) + n-1$

For length 0 and 1, the list doesn't need to be sorted. So, the runtime is 1.

Then, $E(T(n)) = \frac{2}{n}\sum_{q=2}^{n-1} E(T(q)) + (\frac{4}{n} + n - 1)$

$$\lim_{n\to\infty} \frac{\frac{4}{n} + n - 1}{n} = \lim_{n\to\infty} \frac{4}{n^2} + 1 - \frac{1}{n} = 1$$

Then, $\frac{4}{n} + n - 1 = \Theta(n)$

Therefore, $E(T(n)) = \frac{2}{n}\sum_{q=2}^{n-1} E(T(q)) + \Theta(n)$

3. Show that

$$\sum_{k=2}^{n-1} k \lg k \leq \frac{1}{2} n^2 \lg n - \frac{1}{8} n^2.$$

$$\text{LHS} = \sum_{K=2}^{\lceil \frac{n}{2} \rceil - 1} K \log K + \sum_{K=\lceil \frac{n}{2} \rceil}^{n-1} K \log K$$

$$\leq \sum_{K=2}^{\frac{n}{2}} K \log K + \sum_{K=\frac{n}{2}+1}^{n-1} K \log K$$

$$\leq \sum_{K=2}^{\frac{n}{2}} K \log(\tfrac{n}{2}) + \sum_{K=\frac{n}{2}+1}^{n} K \log(n)$$

$$= \log(\tfrac{n}{2}) \sum_{K=2}^{\frac{n}{2}} K + \log n \sum_{K=\frac{n}{2}+1}^{n} K$$

$$= (\log n - \log 2) \frac{\frac{n}{2} \times (\frac{n}{2}+1)}{2} + \log n \left(\frac{n(n+1)}{2} - \frac{(\frac{n}{2})(\frac{n}{2}+1)}{2} \right)$$

$$= -\log 2 \times \frac{\frac{n}{2}(\frac{n}{2}+1)}{2} + \log n \left(\frac{n(n+1)}{2} - \frac{(\frac{n}{2})(\frac{n}{2}+1)}{2} + \frac{\frac{n}{2}(\frac{n}{2}+1)}{2} \right)$$

$$= -\log 2 \times \frac{n(n+2)}{8} + \log n \times \frac{n(n+1)}{2}$$

$$= \frac{1}{2} \times n \log n \times (n+1) - \frac{1}{8} \times n(n+2) \log 2$$

$$\leq \frac{1}{2} \times n \log n \times n - \frac{1}{8} \times n \times n$$

$$= \frac{1}{2} n^2 \log n - \frac{1}{8} n^2$$

Therefore, $\sum_{K=2}^{n-1} K \log K \leq \frac{1}{2} n^2 \log n - \frac{1}{8} n^2$

4. We know. $E(T(n)) = \frac{2}{n}\sum_{q=2}^{n-1} E(T(q)) + \Theta(n)$ (8.1)

$\sum_{k=2}^{n-1} k\log k \le \frac{1}{2}n^2\log n - \frac{1}{8}n^2$ (8.2)

Prove: $E(T(n)) = \Theta(n\log n)$

$E(T(n)) = \frac{2}{n}\sum_{q=2}^{n-1} E(T(q)) + \Theta(n)$ (8.1)

Suppose $E(T(n)) \le K \cdot n\log n$

Then, (8.1) $\rightarrow E(T(n)) \le \frac{2}{n}\sum_{q=2}^{n-1} K \cdot q\log q + \Theta(n)$

$= \frac{2}{n} \times K \left(\sum_{q=2}^{n-1} q\log q\right) + \Theta(n)$

We know (8.2) $\sum_{k=2}^{n-1} k\log k \le \frac{1}{2}n^2\log n - \frac{1}{8}n^2$

Then, $E(T(n)) \le \frac{2K}{n} \times \left(\frac{1}{2}n^2\log n - \frac{1}{8}n^2\right) + \Theta(n)$

$= Kn\log n - \frac{Kn}{4} + \Theta(n)$

$\le Kn\log n$

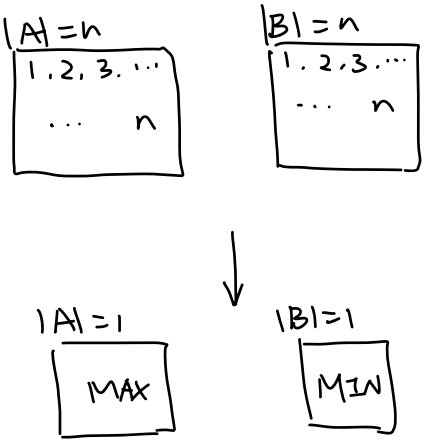
Therefore, $E(T(n)) = \Theta(n\log n)$

Problem 9

At the beginning, all n elements are candidates for max and min but at the end, we only keeps 1 max and 1 min, which via comparisons. 2n is reduced to 2.

Let set $A = B = \{x_1, x_2, \cdots, x_n\}$. We want to have $A = \{Max\}$ and $B = \{Min\}$. The upper bound is $2n-2$ since we want to reduce the $|A|+|B|$'s value from $2n$ to n. For the lower bound, we consider different cases when comparing 2 elements, x and y.

1. If $x \in A$, $y \in A$, $x \in B$, and $y \in B$, then we can deduct 2 elements from set A and B. For example, if $x > y$, then $x \notin B$ and $y \notin A$. If $x = y$, it is also safe to make $x \notin B$ and $y \notin A$.
2. For all the other cases, only 1 element can be deducted from A or B. For example, if $x \in A$, $x \in B$, $y \in A$, and $y \notin B$. after comparison, if $x > y$, $y \notin A$; else if $y \geq x$, $x \notin A$. The other example is $x \in A$ and $y \in A$. When $x > y$, $y \notin A$.



Then, we count the number of cases for 1 since it helps speed up the process. Since there are at most $\lfloor \frac{n}{2} \rfloor$ disjoint pairs,

$$\#\text{ comparisons} \geq n-2-\lfloor \frac{n}{2} \rfloor$$

$$\#\text{ comparisons} \geq \lfloor \frac{4n-n}{2} \rfloor - 2$$

$$\#\text{ comparisons} \geq \lfloor \frac{3n}{2} \rfloor - 2$$

Problem 10

a) Suppose there are K elements smaller than weighted median X_K

Then, $\sum_{x_i<x_K} w_i = \frac{k}{n}$ and $\sum_{x_i>x_K} w_i = \frac{n-k-1}{n}$ if $w_i = \frac{1}{n}$ for $i = 1, \cdots, n$.

We have $\frac{k}{n} < \frac{1}{2}$ and $\frac{n-k-1}{n} \leq \frac{1}{2}$

$\Rightarrow \quad k < \frac{n}{2}$ and $2n - 2k - 2 \leq n$

$\Rightarrow \quad k < \frac{n}{2}$ and $k \geq \frac{n}{2} - 1$

$\Rightarrow \quad \frac{n}{2} - 1 \leq k < \frac{n}{2}$

$\Rightarrow \quad k = \lceil \frac{n}{2} \rceil - 1$

Therefore, X_K is also the median of $x_1, x_2, \cdots, x_n$

(b) 1. Sort the list in the increasing order based on the values using merge sort. $\rightarrow O(n \log n)$

2. Declare a variable calculating the current weight sum S. $\rightarrow O(1)$

3. For x in range(n): } $O(n)$

    S += weight(x)

    if S $\geq \frac{1}{2}$:

        return list(x)

Runtime = $O(n \log n) + O(1) + O(n) = O(n \log n)$

(c) This can be achived by using the updated version of SELECT.

Let X_K be the median of medians of $x_1, x_2, \cdots, x_n$.

Then, compute the value of $\sum_{x_i<x_K} w_i$ and $\sum_{x_i>x_K} w_i$.

If one of them is greater than $\frac{1}{2}$,

recurse call on the sublist with the greater half or the smaller half depends on which one is greater than $\frac{1}{2}$. e.g. if $\sum_{x_i<x_K} w_i > \frac{1}{2}$, recurse on the smaller half.

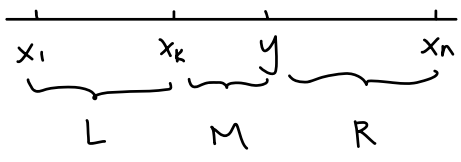
else, stop and return X_K to be the weighted median.

Runtime remains to be $\Theta(n)$ with the modifications.

d) Let x_k be the weighted median. and $A = \sum_{i=1}^{n} w_i \, d(x_k, x_i)$

Prove by contradiction:

Supposed there exists $y \neq x_k$ where y minimizes $\sum_{i=1}^{n} w_i \, d(y, x_i) = B$.



WLOG. suppose $y > x_k$. Let $\{x \mid x < x_k\} \in L$,

$\{x \mid x_k \leq x \leq y\} \in M$ and $\{x \mid y < x < x_n\} \in R$.

$$B - A = \sum_{x_i \in L} w_i (y - x_k) + \sum_{x_i \in R} w_i (x_k - y) + \sum_{x_i \in M} \left((y - x_i) - (x_i - x_k) \right)$$

$$= y \left(\sum_{x_i \in L} w_i - \sum_{x_i \in R} w_i + \sum_{x_i \in M} w_i \right) + x \underbrace{\left(\sum_{x_i \in R} w_i - \sum_{x_i \in L} w_i + \sum_{x_i \in M} w_i \right)}_{\downarrow 0} - 2 \sum_{x_i \in M} w_i x_i$$

Since x_k is the weighted median, $\sum_{x_i \in L} w_i < \frac{1}{2}$ and $\sum_{x_i \in M \cup R} w_i \leq \frac{1}{2}$.

$$\sum_{x_i \in L} w_i - \sum_{x_i \in R} w_i + \sum_{x_i \in M} w_i \geq 2 \sum_{x_i \in M} w_i$$

$$B - A \geq y \times 2 \sum_{x_i \in M} w_i - 2 \sum_{x_i \in M} w_i x_i$$

$$= 2 \sum_{x_i \in M} w_i (y - x_i)$$

Since $y > x_i$, $B - A > 0$. Then, $B > A$. Thus, the sum given by y is greater, which gives contradiction. Therefore, the weighted median is a best solution. ■

(e) The weighted distance is now $d(a, b) = w_{ab} \cdot (|a_x - b_x| + |a_y - b_y|)$

$$= w_{ab} |a_x - b_x| + w_{ab} \cdot |a_y - b_y|$$

Then, the sum will be $\sum_{i=1}^{n} w_i \, d(P, P_i)$ where $P = (P_x, P_y)$

$$= \sum_{i=1}^{n} w_i \left(|P_x - x_i| + |P_y - y_i| \right)$$

$$= \sum_{i=1}^{n} w_i |P_x - x_i| + \sum_{i=1}^{n} w_i |P_y - y_i|$$

In this way, we can treat the 2-dim distance as a sum of two 1-dim distances As proved in the previous part, for 1-dim, the best solution is the weighted median. Thus, for 2-dim distance, the best solution is to have the x-coordinate value to be the weighted median of x_i and the -coordinate value to be the weighted median of y_i.