- # Question 3

  We need to find its median and optimally remove/add 1s to other numbers to bring them to the median for each subsequence. The number of 1s added or removed will be the cost of that particular subsequence. We also need to maintain a BST to perform insertion and deletion in logn.

  ## ***Pseudocode:***

```
struct Node (val, l, r, size) {}
Balance (root node) {
        A[size of root] ← 0;
        preorder traversal to fill A;
        Create the tree using divide and conquer;
        return root;
}
Insert_Node (root, value)    {insert nodes in a way to keep it balanced}
Maximum_Value(node)     {find the maximum value in the corresponding BST in O(logn)}
Minimum_Value(node)      {find the minimum value in the corresponding BST in O(logn)}
Push_Value(value, k, l, h){
        ma ← Maximum_Value(l);
        if(m <= value){
                Insert_Node(l, value); S1 ← S1+ value;
                if(size(h) >= k/2){
                        mi ← Minimum_Value(h); Insert_Node(l, mi);
                        S1 ← S1+ mi; Delete(Search(mi from h)); S2 ← S2 - mi;
                }
        }else{
                Insert_Node(l, value); S1 ← S1+ value;
                if(size(l) >= k/2){
                        ma = Maximum_Value(l);Insert_Node(h, ma);
                        S2 ← S2 + ma; Delete(Search(ma from l));S1 ← S1 - ma;
                }
        }
}
Pop_Value(value, k, l, h){
        if(Seach(value in l))    {Delete(value from l); S1 ← S1 – value;}
        else     {Delete(value from u);}
        S2 ← S2 – value;
        if(l <> NULL){
                mi ← Minimum_Value(); Insert_Node(l , mi); S1 ← S1 + mi;
                delete(Search(mi from h)); S2 ← S2 – mi;
        }
}
```

```
Min_Cost(A, k){
        mic ← 1e10; S1 ← 0; S2 ← 0;
        root nodes for 2 BST are l and h;
        Insert_Node(l, A[0]); S1 ← A[0];
        for(i=0 to k-1)          { Push_Value(A[i] , k, l, h);}
        med ← 0;
        if(k % 2 <> 1)    {med ← Minimum_Value(l); mic ← min(mic , S2 – S1 + med);}
        for (i = k to A.size() - 1) {
                Pop_Value(A[i - k] , k); Push_Value(A[i] , k); med ← 0;
                if(k % 2 <> 1)          { med ← Maximum_Value(l);}
                mic ← min(mic, S2 – S1 + med);
        }
}
```

Proof of correctness:

It will be sufficient for this problem to prove that we need to find the median to minimise the cost. This could be reframed as the median minimises the sum of absolute deviations.

We basically want to minimize $\Sigma|s_i - x|$, sigma over 1 to N and s_i are the elements of the array, with x being the variable we want to find. We differentiate the function and equate t to zero in order to minimize it. We intuitively see that $\Sigma|s_i - x|$ for could be written as $E(|s_i - x|)$, E is expectation and x is a random variable. Now, the differential of $E(|s_i - x|)$

$=E(\text{differential of } |s_i - x|)$

$=E((-1)(s_i - x)/|s_i - x|)$

$=E(\{s_i < x\} - \{s_i > x\})$

$=P(s_i < x) - P(s_i < x)$ should be equal to 0

Therefore $P(s_i < x) = P(s_i < x) = 0.5$

The number of positive sums $(s_i - x)$ becomes equal to negative sums. This could only happen when we take x as the median of the sample elements.

Hence Proved.

- # Question 4

We find the maximum common subsequence between S and S'. After this, the minimum number of added elements will be the number of elements in S' - length of maximum common subsequence.

Pseudocode:

```
Solve(S, S'){
        x = S.size(); y = S'.size(); maximum = 0;
        for(i = 0 to y - 1){
                z = 0; cnt = 0;
                for(j = i to x - 1){
                        While(S[z] != S'[j] and z < x)   {z++;}
                        if(z < x)          {cnt = cnt + 1;}
                        else break;
                }
                maximum = max(maximum, cnt);
        }
        return (y - maximum);
}
```

Time Complexity:

The inner loop runs for x+y times Since we can find the maximum sequence in O(n+m). The outer for loop plans for y times. Therefore, the time complexity is O(m(m+n)).

Proof of correctness:

For the base case, let's assume the size of S' to be zero, then our algorithm returns 0, which is true. Now let's take the (i-1)th iteration to be accurate, and we need to prove the ith iteration. If S'[i] does not match S[z], we increase z by one, or if it matches then we increment z, count and move to the next iteration. We keep a check on z to break the loop when it exceeds x. The maximum of the previous answer and current count ensures that the algorithm always pics the maximum common subsequence length.

When we iterate over all of the array S, we check for common elements for prefix and suffix and subtract them from S'. This number gives the number of elements that we need to add in the beginning and end of S, so that S' becomes a subsequence of S.