J192 - BIGGER, BETTER

PROBLEM IDEA BY ALEX POON

PROBLEM SET BY CHARLIE LI

AGENDA

- 1. Problem statement & Statistics
- 2. Solution for subtask 1
- 3. Solution for K is odd
- 4. General Solution

PROBLEM STATEMENT

PROBLEM BACKGROUND

- 1. Given N, K and an array with N integers
- 2. Choose K integers among them
- 3. Substitute that K integers by the median of them
- 4. Find the maximum total sum of the array can be attained
- 5. Output the K integers you choose

PROBLEM BACKGROUND

- Example: N = 7, K = 3, array = {1, 3, 4, 8, 9, 10, 13}
- You may choose {1, 4, 8} → {4, 4, 4} → {3, 4, 4, 4, 9, 10, 13}, sum = 47
- Or you may choose {1, 8, 13} → {8, 8, 8} → {3, 4, 8, 8, 8, 9, 10}, sum = 50
- But the optimal way is {1, 9, 10} → {9, 9, 9} → {3, 4, 8, 9, 9, 9, 13}, sum = 55

- In this sample, sum = 55 is optimal, so you should output 1, 9, 10

PROBLEM BACKGROUND

- Example: N = 4, K = 2, array = {1, 3, 4, 8}
- You may choose {1, 3} → {2, 2} → {2, 2, 4, 8}, sum = 16
- Or you may choose {1, 4} → {2.5, 2.5} → {2.5, 2.5, 3, 8}, sum = 16
- In fact, the optimal sum can be attained is 16
- So, you may output 1, 3 or 1, 4 or other combinations attain 16

PROBLEM SUBTASK

- 18 marks for N <= 100, K = 3
- 27 marks for K <= N <= 4000 and K is odd
- 21 marks for K <= N <= 4000 and K may be even
- 19 marks for K <= N <= 500000 and K is odd
- 15 marks for K <= N <= 500000 and K may be even

STATISTIC AND EXPECTATION

- 12 people get 18 marks (subtask 1)
- 5 people get 45 marks (subtask 1 + 2)
- 1 person gets 64 marks (subtask 1 + 2 + 4) → K is odd
- 1 person gets 85 marks (subtask 1 + 2 + 3 + 4)
- 6 people get full marks
- Out of 54 people attempted (29 people get 0 ☹)

STATISTIC AND EXPECTATION

- Average = 22.037 (highest among 4 tasks)
- Attempt = 54

- It is close to my expectation. However, I expect more people can pass subtask 1 ☹
- Some candidates get WAs because of some minor common mistakes

SOLUTION FOR SUBTASK 1

- IDEA
- SOLUTION SKETCH
- INPLEMENTATION

SOLUTION 1 - IDEA

- Subtask 1 is very special one where K = 3 (which is a very small constant)
- “Pure exhaustion” solution works

- (Exhaustion == brute force == try all combination)

SOLUTION 1 - IDEA

- Try all combination → choose any K integers among all N integers
- E.g. N = 5, K = 3, array = {1, 3, 4, 8, 9}
- Try to choose
  {1, 3, 4}, {1, 3, 8}, {1, 3, 9}, {1, 4, 8}, {1, 4, 9}
  {1, 8, 9}, {3, 4, 8}, {3, 4, 9}, {3, 8, 9}, {4, ,8, 9}

SOLUTION 1- IDEA

- For subtask 1, N <= 100 and K = 3
- Total number of different combination = C(100, 3) = 161700

- Computer can easily try up to 10^7 different combinations in 1 second!
- So this works for subtask 1

SOLUTION 1- SOLUTION SKETCH

1. Iterate any three numbers in the array
2. In each iteration,
    2.1. Find the median of the three selected numbers
    2.2. Find the sum after transforming the three numbers to their median
    2.3. Record the maximum sum can be obtained
    2.4. Also record choosing which three numbers can obtain the maximum sum

SOLUTION 1 – IMPLEMENTATION – STEP 1

- We can use 3 nested for loops to exhaust which 3 numbers to choose

```
for (int i = 1; i <= n; i++)
    for (int j = i + 1; j <= n; j++)
        for (int k = j + 1; j <= n; k++)
```

SOLUTION 1 – IMPLEMENTATION – STEP 2.1

- Then, find the median of the three chosen number

```
sort(a + 1, a + 1 + n);
for (int i = 1; i <= n; i++)
        for (int j = i + 1; j <= n; j++)
                for (int k = j + 1; j <= n; k++) {
                        int median = a[j];
                }
```

As we keep k > j > i, which means a[k] >= a[j] >= a[i] holds if a[] is sorted

Median of a[i], a[j], a[k] is always a[j]

SOLUTION 1 – IMPLEMENTATION – STEP 2.1

- BAD practice to find the median – using if

```
for (int i = 1; i <= n; i++)
        for (int j = i + 1; j <= n; j++)
                for (int k = j + 1; j <= n; k++) {
                        if (a[i] <= a[j] && a[j] <= a[k]) median = a[j];
                        else if (a[k] <= a[j] && a[j] <= a[i]) median = a[j];
                        else if (a[j] <= a[i] && a[i] <= a[k]) median = a[i];
                        else if (a[k] <= a[i] && a[i] <= a[j]) median = a[i];
                        else median = a[k];
                }
```

- Less experienced student may miss some cases

SOLUTION 1 – IMPLEMENTATION – STEP 2.2

- Calculate the sum after transforming
- An easier approach: calculate the change of sum after transforming only

```
sort(a + 1, a + 1 + n);
for (int i = 1; i <= n; i++)
        for (int j = i + 1; j <= n; j++)
                for (int k = j + 1; j <= n; k++) {
                        int median = a[j];
                        int original_sum = a[i] + a[j] + a[k];
                        int new_sum = median * 3;
                        int change = new_sum – original_sum;
                }
```

SOLUTION 1 – IMPLEMENTATION – STEP 2.3

- Record the optimal sum or optimal change of sum

```
sort(a + 1, a + 1 + n);
for (int i = 1; i <= n; i++)
        for (int j = i + 1; j <= n; j++)
                for (int k = j + 1; j <= n; k++) {
                        int change = (a[i] + a[j] + a[k]) - 3 * a[j];
                        if (change > best_sum) best_sum = change;
                }
```

SOLUTION 1 – IMPLEMENTATION – STEP 2.4

- Record the optimal sum or optimal change of sum also the set of numbers we choose

```
sort(a + 1, a + 1 + n);
for (int i = 1; i <= n; i++)
        for (int j = i + 1; j <= n; j++)
                for (int k = j + 1; j <= n; k++) {
                        int change = (a[i] + a[j] + a[k]) - a[j] * 3;
                        if (change > best_sum) {
                                best_sum = change;
                                ans1 = a[i], ans2 = a[j], ans3 = a[k];
                        }
                }
```

SOLUTION 1 – IMPLEMENTATION – LAST STEP

```
best_sum = -1000000000;
sort(a + 1, a + 1 + n);
for (int i = 1; i <= n; i++)
        for (int j = i + 1; j <= n; j++)
                for (int k = j + 1; j <= n; k++) {
                        int change = (a[i] + a[j] + a[k]) - a[j] * 3;
                        if (change > best_sum) {
                                best_sum = change;
                                ans1 = a[i], ans2 = a[j], ans3 = a[k];
                        }
                }
```

SOLUTION 1 – COMMON MISTAKE

- Consider N = K = 3, a = {1, 2, 10},
- The only way to choose 3 integers is {1, 2, 10} → {2, 2, 2},
- Where change of sum is 6 - 13 = -7

- Without best_sum = -1000000000; Your program may not work in this case
- In general, best_sum should be lower than –(N * R) where R is the range of elements

SOLUTION FOR K IS ODD

- IDEA
- SOLUTION SKETCH
- INPLEMENTATION

SOLUTION 2 – IDEA – OBSERVATION 1

- When choosing ODD number of elements
- The median of chosen element must be one of the elements

- N = 5, K is odd, array = {1, 3, 4, 8, 9}
- The possible median that the numbers change to must be 1 or 3 or 4 or 8 or 9
- It is impossible to choose K integers where their median is 2 or 5 or 6 etc

SOLUTION 2 – IDEA

- Let's try fix the median (denote as M) first
- Then construct a set of integers to choose such that the median of them = M

- Array = {1, 3, 4, 8, 9}, K = 3, fix M = 4
- The valid set to choose can be {1, 4, 8}, {3, 4, 8}, {1, 4, 9}, {3, 4, 9}

SOLUTION 2 – IDEA – OBSERVATION 2

- Array = {1, 3, 4, 8, 9}, K = 3, fix M = 4
- The valid set to choose can be {1, 4, 8}, {3, 4, 8}, {1, 4, 9}, {3, 4, 9}

- In general, in order to make median = M = 4, you need to choose:
- M itself
- K/2 other elements which is <= M
- K/2 other elements which is >= M

SOLUTION 2 – IDEA - OBSERVATION 2

- E.g. Array = {1, 3, 4, 8, 9, 10, 13}, K = 5, fix M = 8
- You need to choose:

  8 itself

  2 other integers <= 8 {any of 1, 3, 4}

  2 other integers >= 8 {any of 9, 10, 13}
- In each combination, the median is 8
- E.g. {1, 3, 8, 9, 10} or {3, 4, 8, 9, 13}……

SOLUTION 2 – IDEA – OBSERVATION 3

- Among all set, we would like to find which leads to the largest change of sum
- As all valid set will eventually change to {8, 8, 8, 8, 8}
- {3, 4, 8, 10, 13} → {8, 8, 8, 8, 8}, {1, 4, 8, 10, 13} → {8, 8, 8, 8, 8} ……
- We would choose the smallest possible elements to attains the largest change
- You need to choose:

  8 itself

  2 other smallest integers <= 8 {any of 1, 3, 4}

  2 other smallest integers >= 8 {any of 9, 10, 13}

SOLUTION 2 – IDEA – CONCLUSION

- After fixing median M, the only combination we need to try is:
- You need to choose:

    M itself

    K/2 other smallest integers <= M

    K/2 other smallest integers >= M

SOLUTION 2 – SOLUTION SKETCH

1. Iterate M along the value in a, and try to fix M as the median

   (which means, let M = a[1], M = a[2] … M = a[n])

2. Find the smallest other K/2 elements <= M

   (If a[] is sorted, that K/2 elements are a[1], a[2], a[3] … a[k/2])

3. Find the smallest other K/2 elements >= M

   (If M = a[i], that K/2 elements are a[i+1], a[i+2], a[i+3] … a[i+k/2])

4. Calculate the sum, record the optimal answer

SOLUTION 2 – IMPLEMENTATION – STEP 1

- Iterate the median along a[]

```
for (int i = 1; i <= n; i++) {
        int M = a[i];
}
```

SOLUTION 2 – IMPLEMENTATION – STEP 2, 3

- Find the smallest other K/2 elements <= M & >= M

```
for (int i = 1; i <= n; i++) {
        int M = a[i];
        if (i <= K/2) continue; // as we don't have K/2 other elements <= M
        else original_sum += sumof(1, K/2);
        if (i + K/2 > n) continue; // as we don't have K/2 other elements >= M
        else original_sum += sumof(i+1, i+K/2);
}
```

SOLUTION 2 – IMPLEMENTATION – STEP 2, 3

- To calculate the sum of consecutive elements: use partial sum
- {1, 3, 4, 8, 9, 10, 13} → cumulative sum array b = {1, 4, 8, 16, 25, 35, 48}
- a[l] + a[l+1] + … a[r] == b[r] – b[l – 1]

- E.g. a[3] + a[4] + a[5] = 4 + 8 + 9 = 21 = 25 – 4 = b[5] – b[2]

SOLUTION 2 – IMPLEMENTATION – STEP 2, 3

- Find the smallest other K/2 elements <= M & >= M

```
for (int i = 1; i <= n; i++) {
        int M = a[i];
        if (i <= K/2) continue; // as we don’t have K/2 other elements <= M
        else original_sum += b[k/2] – b[0]
        if (i + K/2 > n) continue; // as we don’t have K/2 other elements >= M
        else original_sum += b[i+K/2] – b[i];
}
```

SOLUTION 2 – IMPLEMENTATION – STEP 4

- Record the optimal answer

```
for (int i = 1; i <= n; i++) {
        int M = a[i];
        if (i <= K/2) continue; // as we don’t have K/2 other elements <= M
        else original_sum += b[k/2] – b[0]
        if (i + K/2 > n) continue; // as we don’t have K/2 other elements >= M
        else original_sum += b[i+K/2] – b[i];
        int change = M * k – original_sum;
        if (change > best_sum) best_sum = change, best_median = i;
}
output a[1] .. a[k/2], a[best_median], a[best_median + 1] .. A[best_median + k/2]
```

SOLUTION 2 – IMPLEMENTATION – STEP 4

- Record the optimal answer

```
for (int i = 1; i <= n; i++) {
        int M = a[i];
        if (i <= K/2) continue; // as we don’t have K/2 other elements <= M
        else original_sum += b[k/2] – b[0]
        if (i + K/2 > n) continue; // as we don’t have K/2 other elements >= M
        else original_sum += b[i+K/2] – b[i];
        int change = M * k – original_sum;
        if (change > best_sum) best_sum = change, best_median = i;
}
output a[1] .. a[k/2], a[best_median], a[best_median + 1] .. A[best_median + k/2]
```

SOLUTION 2 – CONCLUSION

- Time complexity: O(N)
- Works for K is odd only and can pass subtask 1, 2, 4
- An O(N^2) solution (without partial sum) can pass subtask 1, 2 only
- If you forget to use long long or initialize best_sum to be a small enough number (2.5e10 this time), then you can pass subtask 1, 2 only

GENERAL SOLUTION

- IDEA
- SOLUTION SKETCH

SOLUTION 3 – ITERATE MEDIAN FOR EVEN K

- For K is even, median may not be one of those a[i]
- It may be floating point indeed {1, 2, 3, 4} → {2.5, 2.5, 2.5, 2.5}
- But a similar approach works

SOLUTION 3 – IDEA – OBSERVATION 1

- Instead of changing all chosen value of median
- Change the top half value A[ceiling(K/2)], bottom half to A[floor(K/2)] also works

- {1, 4, 5, 7} → {4.5, 4.5, 4.5, 4.5} = {4, 4, 5, 5} !!!
- Or {1, 3, 4, 7, 9, 9} → {5.5, 5.5, 5.5, 5.5, 5.5, 5.5} = {4, 4, 4, 7, 7, 7}
- As they have same sum

SOLUTION 3 – IDEA

ODD CASE

- Fix M
- You need to choose:

  M itself

  K/2 other smallest integers <= M

  K/2 other smallest integers >= M

EVEN CASE

- Fix M1, M2 (M1 <= M2)
- You need to choose

  M1, M2 themselves

  K/2-1 other smallest integer <= M1

  K/2-1 other smallest integer >= M2

SOLUTION 3 – IDEA

- We need two nested loop to iterate M1, M2
- So we only get an O(N^2) solution which works in subtask 3 only at this moment

SOLUTION 3 – IDEA – OBSERVATION 2

- We only need to try M2 = a[i] and M1 = a[i-1]

SOLUTION 3 – IDEA – OBSERVATION 2

- Consider fixing M2 = a[i], we will choose a[i+1], a[i+2] .. a[i+K/2-1]
- And those value will eventually change to M2
- Then we can choose any value <= M2 as M1 and a[1], a[2] … a[K/2-1]
- a[1], a[2] … a[K/2-1] will eventually change to M1

- To attains the largest sum, we would like M1 as large as possible

SOLUTION 3 – SKETCH

1. Iterate M2 along a[]

    In each iteration:

    2.1. Let M1 be a[i – 1]

    2.2. Choose a[1], a[2] … a[k/2-1] and a[i+1], a[i+2] … a[i+k/2-1]

    2.3. Calculate sum by partial sum, record

SOLUTION 3 – IMPLEMENTATION

Leave as exercise

Any question?