# Day 184/180 Heap Problems

## 1: Take Gifts From the Richest Pile

### Approch 1:

```cpp
class Solution {
public:
    long long pickGifts(vector<int>& a, int k) {
        long long ans = 0; // Initialize the answer variable to store the sum of gifts
        for(int i=0;i<k;i++){ // Perform k iterations
            int mx = a[0], idx = 0; // Initialize mx to the first element of a and idx to 0
            for(int j=0;j<a.size();j++){ // Loop through all elements of a
                if(mx < a[j]){ // Check if a[j] is greater than mx
                    mx = a[j]; // Update mx to the greater value
                    idx = j; // Update idx to the index of the greater value
                }
            }
            a[idx] = sqrt(a[idx]); // Replace the element at idx with its square root
        }
        for(int i=0;i<a.size();i++){ // Loop through all elements of a
            ans += (long long)a[i]; // Add each element to the answer
        }
        return ans; // Return the total sum of gifts
    }
};
```

### Approach 2:

```cpp
class Solution {
public:
    long long pickGifts(vector<int>& gifts, int k) {
        // Create a priority queue to store elements in descending order
        priority_queue<int> pq;

        // Push all elements from the 'gifts' vector into the priority queue
        for(auto &i: gifts){
            pq.push(i);
        }

        long long int ans = 0; // Initialize the answer variable to store the sum of gifts
        int x; // Initialize a variable to store the current top element of the priority queue

        // Perform 'k' iterations
        while(k--){
            x = pq.top(); // Get the top element of the priority queue
            pq.pop(); // Remove the top element from the priority queue
            pq.push(sqrt(x)); // Push the square root of the removed element back into the
priority queue
        }

        // Calculate the sum of all elements remaining in the priority queue
        while(!pq.empty()){
            ans += pq.top(); // Add the top element of the priority queue to the answer
            pq.pop(); // Remove the top element from the priority queue
        }

        return ans; // Return the total sum of gifts
    }
};
```

## 2: Make Array Zero by Subtracting Equal Amounts

```cpp
class Solution {
```

```cpp
public:
    int minimumOperations(vector<int>& nums) {
        int cnt = 0; // Initialize a counter to count the minimum operations

        // Sort the 'nums' vector in ascending order
        sort(nums.begin(), nums.end());

        // Iterate through the sorted vector starting from the second element
        for(int i = 1; i < nums.size(); i++){
            // Check if the current element is different from the previous element
            if(nums[i] != nums[i - 1]){
                cnt++; // Increment the counter if the elements are different
            }
        }

        // Check if the first element of the sorted vector is not 0
        if(nums[0] != 0){
            cnt++; // Increment the counter if the first element is not 0
        }

        return cnt; // Return the total count of minimum operations
    }
};
```

## 3: Relative Ranks

```cpp
class Solution {
public:
    vector<string> findRelativeRanks(vector<int>& score) {
        vector<pair<int,int>> v; // Create a vector of pairs to store scores and their
indices
        int n = score.size(); // Get the size of the 'score' vector
        for(int i = 0; i < score.size(); i++){
            v.push_back({score[i], i}); // Push each score and its index into the vector
of pairs
        }
        // Sort the vector of pairs in descending order based on scores
        sort(v.rbegin(), v.rend());

        vector<string> ans(n); // Initialize a vector of strings to store the relative
ranks
        for(int i = 0; i < n; i++){
            if(i == 0){
                ans[v[i].second] = "Gold Medal"; // Assign "Gold Medal" to the
first place
            }
            else if(i == 1){
                ans[v[i].second] = "Silver Medal"; // Assign "Silver Medal" to the
second place
            }
            else if(i == 2){
                ans[v[i].second] = "Bronze Medal"; // Assign "Bronze Medal" to the
third place
            }
            else{
                ans[v[i].second] = to_string(i + 1); // Assign the rank as a
string for the rest of the places
            }
        }

        return ans; // Return the vector of relative ranks
    }
};
```

## 4: Minimum Amount of Time to Fill Cups

```
class Solution {
public:
    int fillCups(vector<int>& a) {
```

```cpp
        int total = 0; // Initialize a variable to store the total number of filled cups

        priority_queue<int> pq; // Create a priority queue to store cup sizes in descending
order

        // Push non-zero cup sizes into the priority queue
        for(auto x : a) {
            if(x) pq.push(x);
        }
        // Iterate until the priority queue is empty
        while(!pq.empty()){
            int x = pq.top(); // Get the largest cup size from the priority queue
            pq.pop(); // Remove the largest cup size

            int y = 0; // Initialize a variable to store the second largest cup size

            // Check if there is another cup size available in the priority queue
            if(!pq.empty()){
                y = pq.top(); // Get the second largest cup size
                pq.pop(); // Remove the second largest cup size from the priority queue
            }
            else{
                total += x; // Add the remaining cup size to the total and exit the loop
                break;
            }

            total++; // Increment the total number of filled cups
            x--, y--; // Decrease the cup sizes after filling

            // Push the updated cup sizes back into the priority queue if they are greater than 0
            if(x > 0) pq.push(x);
            if(y > 0) pq.push(y);
        }
        return total; // Return the total number of filled cups
    }
};
```