Maximum sum combination

BRUTE FORCE :

Push all combination of sum into maxheap and pop out the kth sum

Code :

```
#include <bits/stdc++.h>
vector<int> kMaxSumCombination(vector<int> &a, vector<int> &b, int n, int k){
  // Write your code here.
  priority_queue<int> q;
  for(int i=0;i<n;i++)
  {
    for(int j=0;j<n;j++)
    {
      q.push(a[i]+b[j]);
    }
  }
  int cnt=0;
  vector<int>  ans;
  while(cnt!=k)
  {
    cnt++;
    ans.push_back(q.top());
    q.pop();
  }
  return ans;
}
```

- Time Complexity : O(N^2 log (N^2))
- Space Complexity : O(log k)

Optimized Approach :

firstly sort the array to get maximum elements at end after all we need the numbers that are maximum to get the sum to be maximum. We then make use of maxheap and set ,maxheap basically stores the maximum sum at the top with the indexes of element that adds upto that sum.

Set is used to remove the duplicates by making use of indexes , if the indexes are already present in the set we do not treat it as new sum and thus skip such

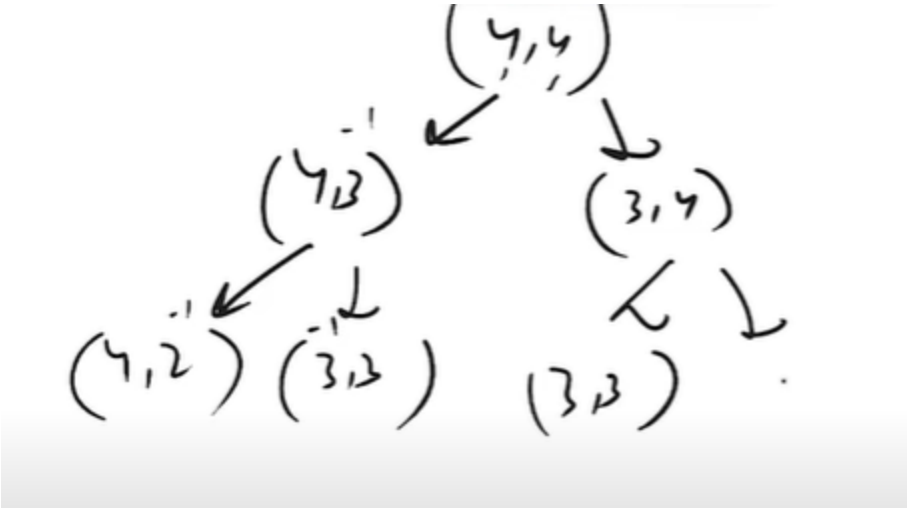
combinations but if it is not present then we add it into the maxheap as well as set.

On each iteration we pop the top element of priority queue and reduce the value of k by adding the top element sum to ans vector. This is repeated until k>0. While pushing into priority queue the elements that can contribute to maximum are a[i-1]+b[j] and a[i]+b[j-1] i.e the cross elements.

1 2 3 4

3 4 5 6

in this example combination can be 4 and 6 then 3 and 6 and 5 and 4 these combination are possible for each index. Thus all combination can be tried out.



```
#include <bits/stdc++.h>
vector<int> kMaxSumCombination(vector<int> &a, vector<int> &b, int n, int k) {
  // Write your code here.
  vector<int> ans;
  sort(a.begin(),a.end());
  sort(b.begin(),b.end());
  priority_queue<pair<int,pair<int,int>>> pq;
  set<pair<int,int>> s;
  s.insert({n-1,n-1});
  pq.push({a[n-1]+b[n-1],{n-1,n-1}});
    while (k>0) {
      auto tp=pq.top();
    int sum=tp.first;
    int i=tp.second.first;
    int j=tp.second.second;
    pq.pop();
    ans.push_back(sum);
    k--;
```

```
    if(i>0 && s.find({i-1,j})==s.end())
    {
      pq.push({a[i-1]+b[j],{i-1,j}});
      s.insert({i-1,j});
    }
    if(j>0 &&s.find({i,j-1})==s.end())
    {
      pq.push({a[i]+b[j-1],{i,j-1}});
      s.insert({i,j-1});
    }
    }
  return ans;
}
```

- Time Complexity : O(k*(log(3*n)+log(set size)))+ O(NlogN)
- Space Complexity : O(2*total unique index sum combination)