## Subsets

Given a set of distinct integers, *nums*, return all possible subsets.

**Note:**

- Elements in a subset must be in non-descending order.
- The solution set must not contain duplicate subsets.

For example,
If ***nums*** = `[1,2,3]`, a solution is:

```
[
  [3],
  [1],
  [2],
  [1,2,3],
  [1,3],
  [2,3],
  [1,2],
  []
]
```

## Solution 1

```
class Solution {
public:
    vector<vector<int> > subsets(vector<int> &S) {
        sort (S.begin(), S.end());
        int elem_num = S.size();
        int subset_num = pow (2, elem_num);
        vector<vector<int> > subset_set (subset_num, vector<int>());
        for (int i = 0; i < elem_num; i++)
            for (int j = 0; j < subset_num; j++)
                if ((j >> i) & 1)
                    subset_set[j].push_back (S[i]);
        return subset_set;
    }
};
```

written by thumike original link here

### Recursive (Backtracking)

This is a typical problem that can be tackled by backtracking. Since backtracking has a more-or-less similar template, so I do not give explanations for this method.

```
class Solution {
public:
    vector<vector<int>> subsets(vector<int>& nums) {
        sort(nums.begin(), nums.end());
        vector<vector<int>> subs;
        vector<int> sub;
        genSubsets(nums, 0, sub, subs);
        return subs;
    }
    void genSubsets(vector<int>& nums, int start, vector<int>& sub, vector<vector
<int>>& subs) {
        subs.push_back(sub);
        for (int i = start; i < nums.size(); i++) {
            sub.push_back(nums[i]);
            genSubsets(nums, i + 1, sub, subs);
            sub.pop_back();
        }
    }
};
```

### Iterative

This problem can also be solved iteratively. Take `[1, 2, 3]` in the problem statement as an example. The process of generating all the subsets is like:

1. Initially: `[[]]`
2. Adding the first number to all the existed subsets: `[[], [1]]`;
3. Adding the second number to all the existed subsets: `[[], [1], [2], [1, 2]]`;
4. Adding the third number to all the existed subsets: `[[], [1], [2], [1, 2], [3], [1, 3], [2, 3], [1, 2, 3]]`.

Have you got the idea :-)

The code is as follows.

```
class Solution {
public:
    vector<vector<int>> subsets(vector<int>& nums) {
        sort(nums.begin(), nums.end());
        vector<vector<int>> subs(1, vector<int>());
        for (int i = 0; i < nums.size(); i++) {
            int n = subs.size();
            for (int j = 0; j < n; j++) {
                subs.push_back(subs[j]);
                subs.back().push_back(nums[i]);
            }
        }
        return subs;
    }
};
```

---

### Bit Manipulation

This is the most clever solution that I have seen. The idea is that to give all the possible subsets, we just need to exhaust all the possible combinations of the numbers. And each number has only two possibilities: either in or not in a subset. And this can be represented using a bit.

There is also another a way to visualize this idea. That is, if we use the above example, `1` appears once in every two consecutive subsets, `2` appears twice in every four consecutive subsets, and `3` appears four times in every eight subsets, shown in the following (initially the `8` subsets are all empty):

`[], [], [], [], [], [], [], []`

`[], [1], [], [1], [], [1], [], [1]`

`[], [1], [2], [1, 2], [], [1], [2], [1, 2]`

`[], [1], [2], [1, 2], [3], [1, 3], [2, 3], [1, 2, 3]`

The code is as follows.

```
class Solution {
public:
    vector<vector<int>> subsets(vector<int>& nums) {
        sort(nums.begin(), nums.end());
        int num_subset = pow(2, nums.size());
        vector<vector<int> > res(num_subset, vector<int>());
        for (int i = 0; i < nums.size(); i++)
            for (int j = 0; j < num_subset; j++)
                if ((j >> i) & 1)
                    res[j].push_back(nums[i]);
        return res;
    }
};
```

---

Well, just a final remark. For Python programmers, this may be an easy task in practice since the `itertools` package has a function `combinations` for it :-)

written by jianchao.li.fighter original link here

## Solution 3

```
class Solution {
public:
vector<vector<int> > subsets(vector<int> &S) {
    vector<vector<int> > res(1, vector<int>());
    sort(S.begin(), S.end());

    for (int i = 0; i < S.size(); i++) {
        int n = res.size();
        for (int j = 0; j < n; j++) {
            res.push_back(res[j]);
            res.back().push_back(S[i]);
        }
    }

    return res;
  }
};
```

written by jaewoo original link here