78. Subsets

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

Note: The solution set must not contain duplicate subsets.

Example:

```
Input: nums = [1,2,3]
Output:
[
  [3],
  [1],
  [2],
  [1,2,3],
  [1,3],
  [2,3],
  [1,2],
  []
]
```

Analysis:

Nums = [1, 2, 3]

List = [[]];

➡ Start = 0, i = 0;
Templist = [1];
list = [[], [1]];

➡ start = 1, i = 1;
templist = [1, 2];
list = [[], [1], [1, 2]];

➡ start = 2, i = 2;
templist = [1, 2, 3];
list = [[], [1], [1, 2], [1, 2, 3]];

⬅ templist.remove = [1, 2];

templist.remove = [1];

➡ start = 1, i = 2;
templist = [1, 3];
list = [[], [1], [1, 2], [1, 2, 3], [1, 3]];

⬅ templist.remove = [1];

templist.remove = [];

➡ start = 0, i = 1;
templist = [2];
list = [[], [1], [1, 2], [1, 2, 3], [1, 3], [2]];

➡ start = i+1 = 2, i = 2;
templist = [2, 3];
list = [[], [1], [1, 2], [1, 2, 3], [1, 3], [2], [2, 3]];

⬅ templist.remove = [2];

⬅ templist.remove = [];

➡ start = 0, i = 2;
templist = [3];
list = [[], [1], [1, 2], [1, 2, 3], [1, 3], [2], [2, 3], [3]];

⬅ templist = [];

Coding in Java:

```java
public class Solution {
    public List<List<Integer>> subsets(int[] nums) {
        List<List<Integer>> list = new ArrayList<>();
        Arrays.sort(nums);

        backtrack(list, new ArrayList<>(), nums, 0);

        return list;
    }


    void backtrack(List<List<Integer>> list, List<Integer> templist, int[] nums, int start) {
        list.add(new ArrayList<>(templist));

        for (int i = start; i < nums.length; i++) {
            templist.add(nums[i]);
            backtrack(list, templist, nums, i+1);
            templist.remove(templist.size() - 1);
        }
    }

}
```

Complexity analysis:

--- Time complexity:

--- Space complexity:

90. Subsets II

Given a collection of integers that might contain duplicates, *nums*, return all possible subsets.

Note: The solution set must not contain duplicate subsets.

Analysis:

--- Same with subsets, except: At a position if *I* > *start* and *nums[i] = nums[i-1] = A*, that means A is already added to the templist. So, we should skip.

Coding in Java:

```java
public class Solution {
    public List<List<Integer>> subsetsWithDup(int[] nums) {
        List<List<Integer>> list = new ArrayList<>();
        Arrays.sort(nums);

        backtrack(list, new ArrayList<>(), nums, 0);

        return list;
    }

    void backtrack(List<List<Integer>> list, List<Integer> templist, int[] nums, int start) {
        list.add(new ArrayList<>(templist));

        for (int i = start; i < nums.length; i++) {
            if (i > start && nums[i] == nums[i-1]) {
                continue;
            }

            templist.add(nums[i]);
            backtrack(list, templist, nums, i+1);
            templist.remove(templist.size() - 1);
        }
    }

}
```

Complexity analysis:

--- Time complexity:

--- Space complexity:

46. Permutations

Given a collection of distinct integers, return all possible permutations.

Example:

```
Input: [1,2,3]
Output:
[
  [1,2,3],
  [1,3,2],
  [2,1,3],
  [2,3,1],
  [3,1,2],
  [3,2,1]
]
```

Analysis:

Nums = [1, 2, 3]

i = 0 --- [1]

i = 0 – skip
i = 1 --- [1, 2]

i = 0 – skip
i = 1 – skip
i = 2 --- [1, 2, 3]

[[1, 2, 3]]

Remove = [1, 2]

Remove = [1]
i = 2 --- [1, 3]

i = 0 – skip
i = 1 --- [1, 3, 2]

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

Remove = [1, 3]
i = 2 – skip

remove = [1]

remove = []

i = 1 --- [2]

i = 0 --- [2, 1]

i = 0 – skip
i = 1 – skip
i = 2 --- [2, 1, 3]

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

Remove = [2, 1]

Remove = [2]

i = 1 --- skip
i = 2 --- [2, 3]

i = 0 --- [2, 3, 1]

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

Remove = [2, 3]

i = 1 – skip
i = 2 – skip

remove = [2]

remove = []

i = 2 --- [3]

i = 0 --- [3, 1]

i = 0 – skip
i = 1 --- [3, 1, 2]

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

remove = [3, 1]
i = 2 --- skip

remove = [3]

i = 1 --- [3, 2]

i = 0 --- [3, 2, 1]

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

Coding in Java:

```java
class Solution {
    public List<List<Integer>> permute(int[] nums) {
        List<List<Integer>> list = new ArrayList<>();

        backtrack(list, new ArrayList<>(), nums);

        return list;
    }


    void backtrack(List<List<Integer>> list, List<Integer> templist, int[] nums) {
        if (templist.size() == nums.length) {
            list.add(new ArrayList(templist));
        }

        for (int i = 0; i < nums.length; i++) {
            if (templist.contains(nums[i])) {
                continue;
            }

            templist.add(nums[i]);
            backtrack(list, templist, nums);
            templist.remove(templist.size() - 1);
        }
    }

}
```

Complexity analysis:

--- Time complexity:

--- Space complexity: