Lecture 19 – Backtracking, Sorting

1. <https://leetcode.com/problems/combination-sum/>

class Solution {

void allCombinationSum(vector<int>&candidates, int index, int target, vector<int>&path, vector<vector<int>>&res)

{

// base case

if(index==candidates.size())

{

if(target==0)

{

res.push\_back(path);

}

return;

}

if(candidates[index]<=target)

{

path.push\_back(candidates[index]);

allCombinationSum(candidates, index, target-candidates[index], path, res); // including

path.pop\_back();

}

// excluding

allCombinationSum(candidates, index+1, target, path, res);

}

public:

vector<vector<int>> combinationSum(vector<int>& candidates, int target) {

vector<vector<int>> res;

vector<int> path;

allCombinationSum(candidates, 0, target, path, res);

return res;

}

};

1. Merge sort
2. #include <iostream>
3. using namespace std;
4. void merge(int arr[], int s, int e)
5. {
6. int mid = (s+e)/2;
7. int i = s;
8. int j = mid+1;
9. int k = s;
10. int temp[100];
11. while(i<=mid && j<=e)
12. {
13. if(arr[i] < arr[j])
14. {
15. temp[k++] = arr[i++];
16. }
17. else
18. {
19. temp[k++] = arr[j++];
20. }
21. }
22. while(i<=mid)
23. {
24. temp[k++] = arr[i++];
25. }
26. while(j<=e)
27. {
28. temp[k++] = arr[j++];
29. }
30. for(int i=s; i<=e; i++)
31. {
32. arr[i] = temp[i];
33. }
34. }
35. void mergeSort(int arr[], int s, int e)
36. {
37. // base case
38. if(s>=e)
39. {
40. return;
41. }
42. // 3 steps
43. int mid = (s+e)/2; // divide
44. mergeSort(arr, s, mid); // Recursively sort
45. mergeSort(arr, mid+1, e);
46. merge(arr, s, e); // merge
47. }
48. int main() {
49. int arr[100];
50. int n;
51. cin>>n;
52. for(int i=0; i<n; i++)
53. {
54. cin>>arr[i];
55. }
56. mergeSort(arr, 0, n-1);
57. for(int i=0; i<n; i++)
58. {
59. cout<<arr[i]<<",";
60. }
61. return 0;
62. }