Problem Solving:

Sorting

Mar. 2019

Honguk Woo

Q : Sum of pair

- Given an array A[] and a number x, check for pair in A[] with sum as x
- A = {1, 4, 45, 6, 10, -8} and sum = 16

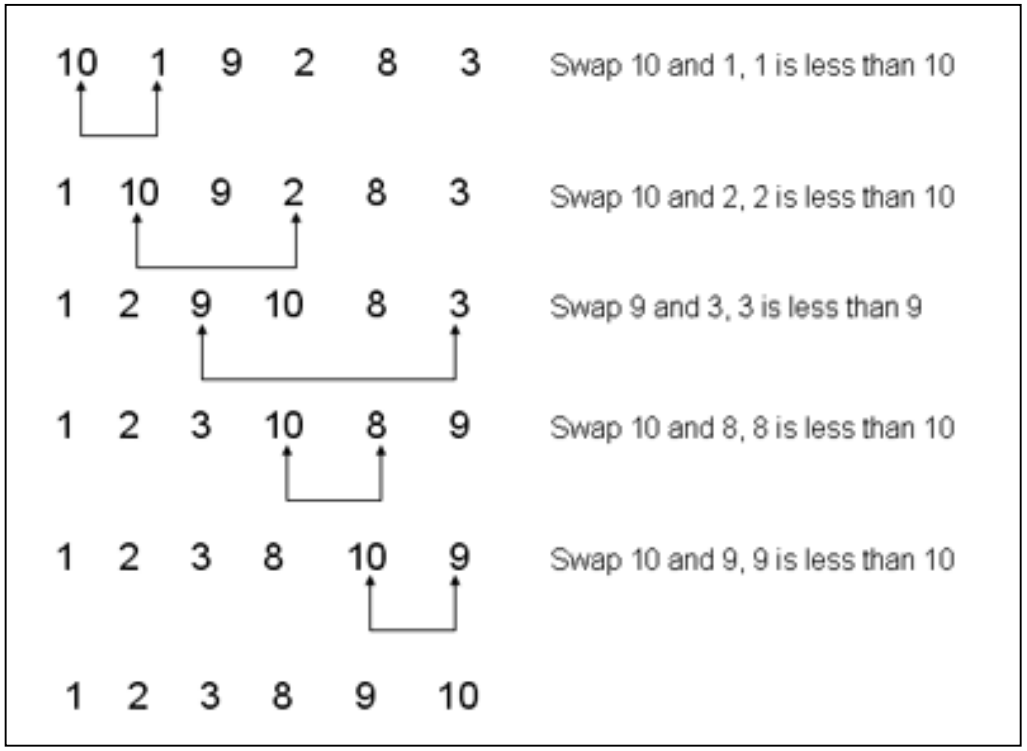
- Complexity of Exhaustive Search ?
- Better solution ?

Sorting

- The most basic technique in programming
- So many solutions for it
  - $O(n^2)$, $O(n \lg n)$, $O(n)$
  - depending on
    - simplicity of mind
    - complexity of insert operation
- Algorithms
  - Selection, Insertion sort
  - Merge sort, Quick sort
  - Counting Sort
  - Bucket Sort

Selection Sort

- Maintain two parts : sorted, unsorted
- Selection : in every iteration i, select the minimum (maximum) from the unsorted part, and move it to the sorted part (current index i)
- Two for-loops:
  - i = 0….n-2
  - J = i+1…n-1
  - Keep the index of minimum
- Comparisons
  - (n - 1) + (n - 2) + … + 1 = n(n - 1)/2



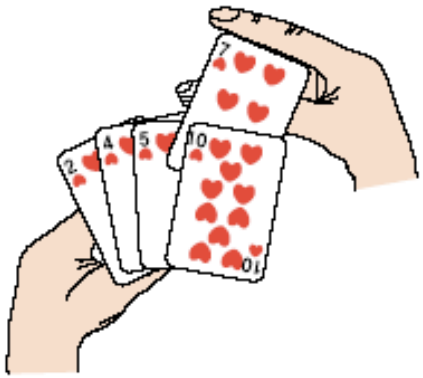
Selection Sort – the code

```
selection_sort(int s[], int n)
{
        int i,j;                    /* counters */
        int min;                    /* index of minimum */

        for (i=0; i<n; i++) {
                min=i;
                for (j=i+1; j<n; j++)
                        if (s[j] < s[min]) min=j;
                swap(&s[i],&s[min]);
        }
}
```

Insertion Sort

- Maintain two parts : sorted, unsorted
- Insertion : in every iteration i, pick i's element and insert it on the sorted part
- Two loops:
  - for-loop: i=1…..n-1
  - while-loop: j=i-1, …. 0 and
    i's value < j's value



54	26	93	17	77	31	44	55	20	Assume 54 is a sorted list of 1 item
26	54	93	17	77	31	44	55	20	inserted 26
26	54	93	17	77	31	44	55	20	inserted 93
17	26	54	93	77	31	44	55	20	inserted 17
17	26	54	77	93	31	44	55	20	inserted 77
17	26	31	54	77	93	44	55	20	inserted 31
17	26	31	44	54	77	93	55	20	inserted 44
17	26	31	44	54	55	77	93	20	inserted 55
17	20	26	31	44	54	55	77	93	inserted 20

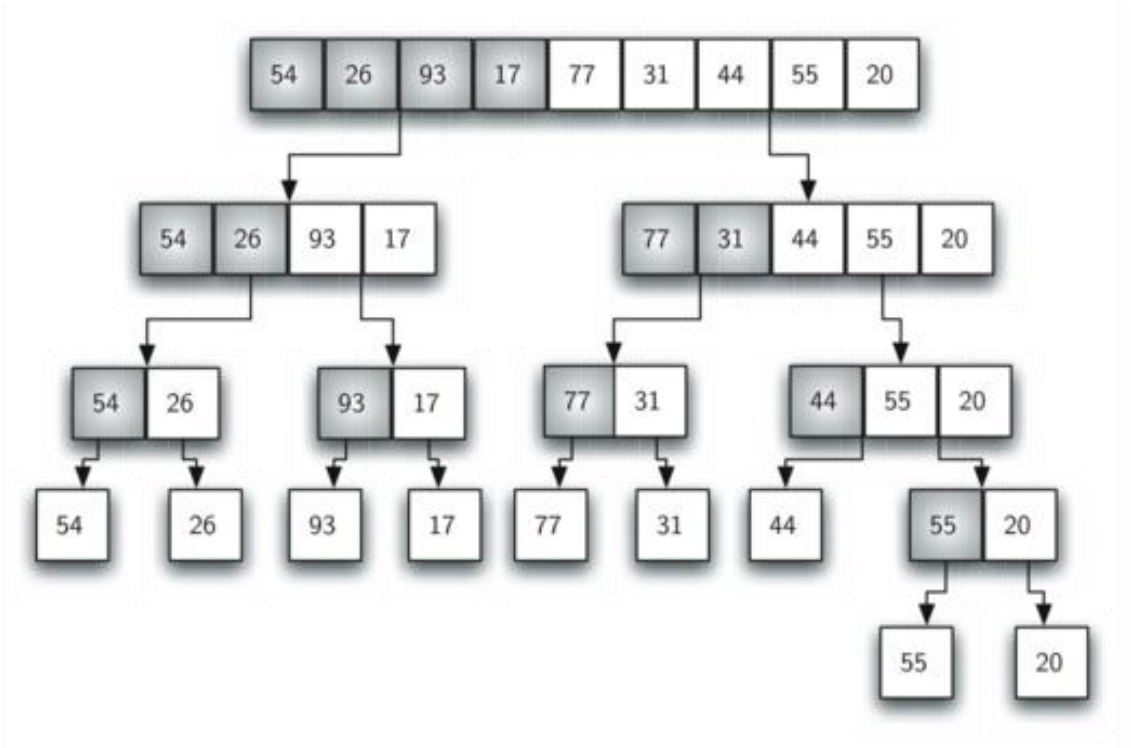
Insertion Sort – the code

```
void insertionSort(int arr[], int n) {
  int i, key, j;
  for (i = 1; i < n; i++) {
    key = arr[i];
    j = i - 1;

    while (j >= 0 && arr[j] > key) {
      arr[j + 1] = arr[j];
      j = j - 1;
    }
    arr[j + 1] = key;
  }
}
```

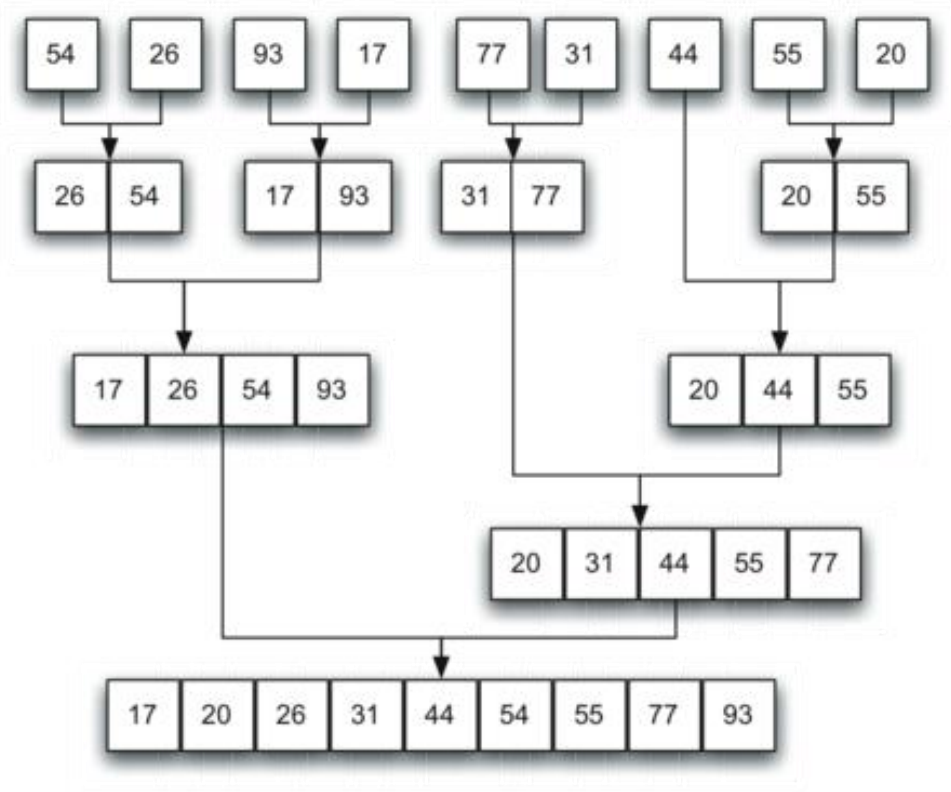
Merge sort – Divide & Conquer

- Divide



Merge sort – Divide & Conquer

- Conquer (merge)



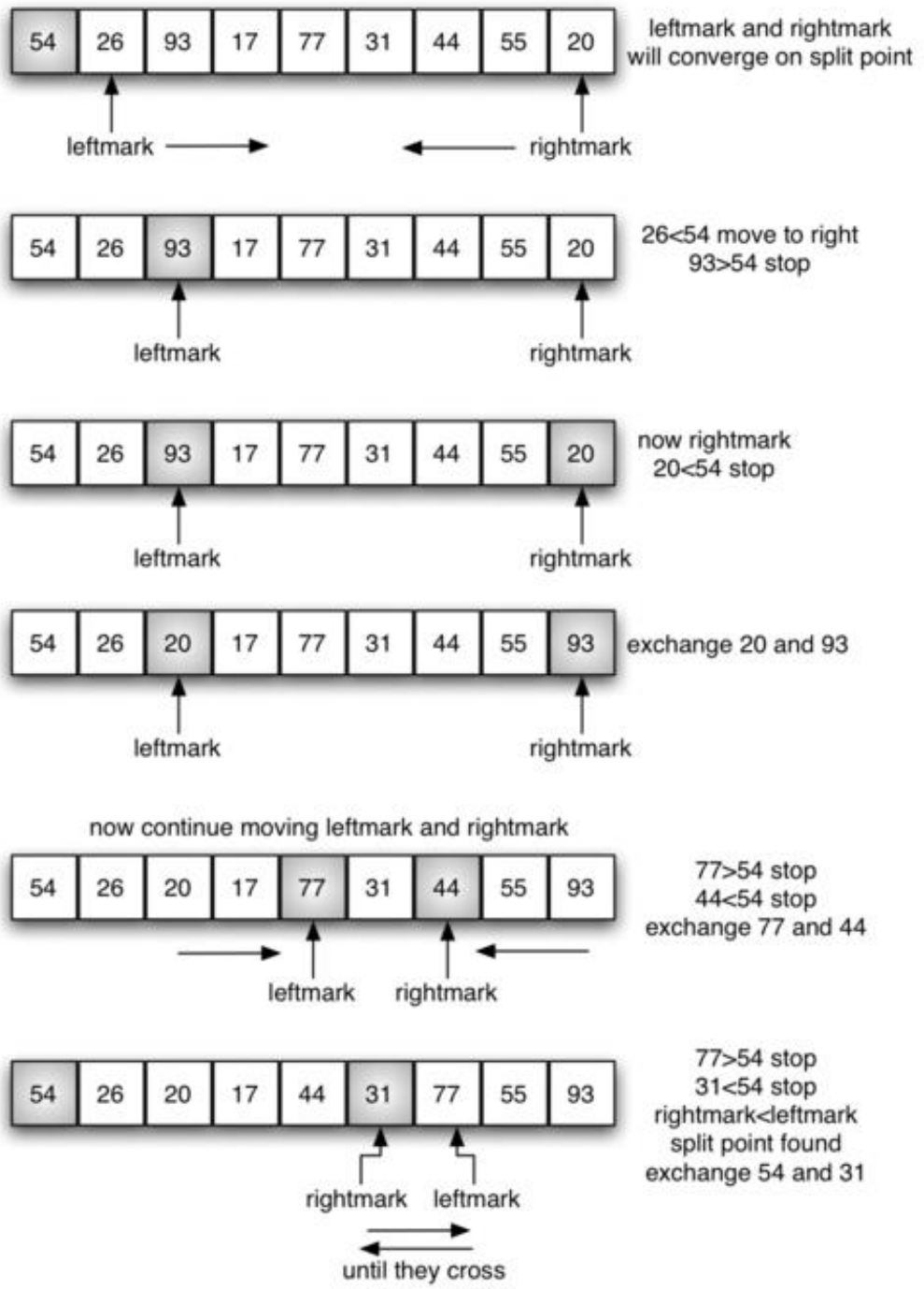
Quick Sort

- Divide-and-Conquer
  - **Divide** the array into two parts
    - the value of x is called pivot

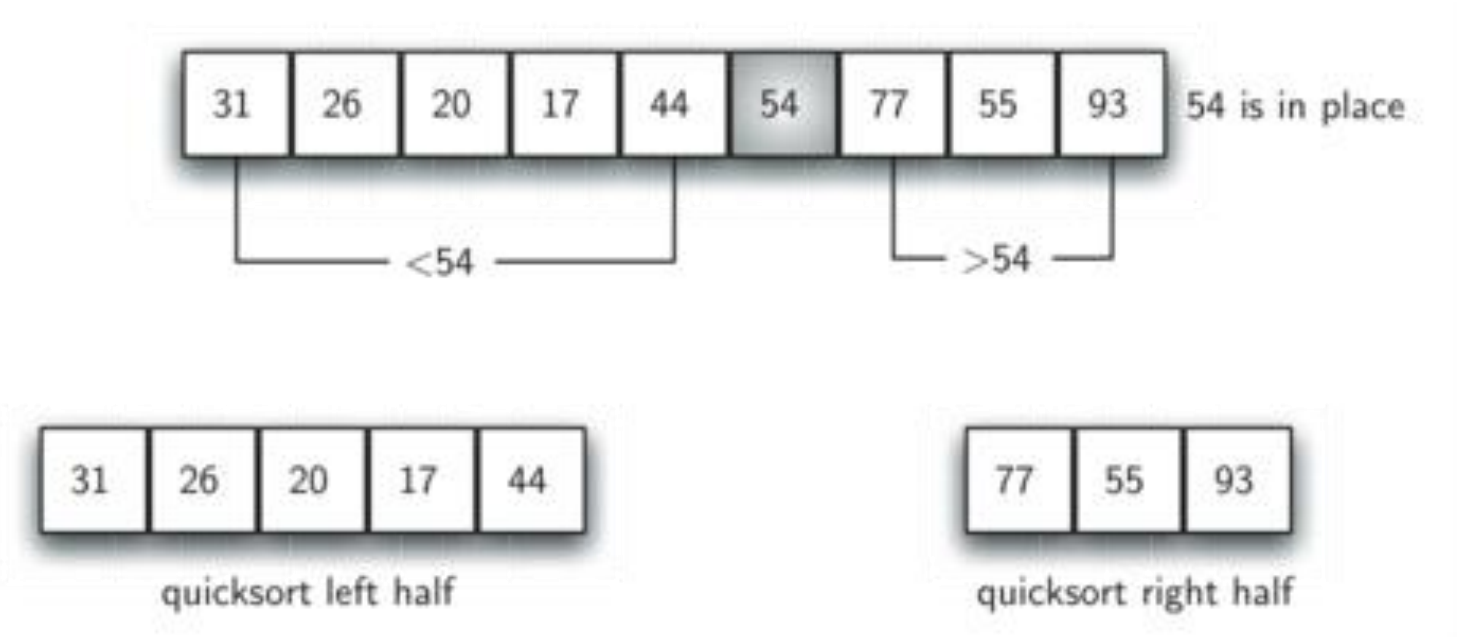
  - **Conquer**
    - do the same to each divided subarray
    - Combine

partition

- pivot: 54



partition


31
26
20
17
44
54
77
55
93
54 is in place
<54
>54
31
26
20
17
44
quicksort left half
77
55
93
quicksort right half

```
void quicksort(int a[], int l, int h)
{
    int p;
    if((h-l)>0) {
        p = partition(a, l, h);
        quicksort(a, l, p-1);
        quicksort(a, p+1, h);
    }
}
```

```
int partition(int a[], int first, int last) {
    int pivot, left, right;
    pivot = first;
    left = first;
    right = last;
    while (left < right) {
       while(a[left] <= a[pivot] && left < last)
         left++;
       while(a[right] > a[pivot] && right > first)
         right--;
       if(left < right) {
         Swap(&a[left], &a[right]);
       }
    }
    Swap(&a[pivot], &a[right]);
    return right;
}
```

Library

```
#include <stdlib.h>

void qsort(void *base, size_t  nel, size_t  width,
        int (*compare) (const void *, const void *));
```

- array: base
- number of elements: nel
- size of each element: width bytes
- you should provide compare function

```
int mycompare(int *i, int *j)
{
        if (*i > *j) return (1);
        elseif (*i < *j) return (-1);
        else return (0);
}
```

```
qsort((char *) a, cnt, sizeof(int), mycompare);
```

Vito's Family

The input consists of several test cases. The first line contains the number of test cases.

For each test case you will be given the integer number of relatives r $(0 < r < 500)$ and the street numbers (also integers) $s_1, s_2, \ldots, s_i, \ldots, s_r$ where they live $(0 < s_i < 30,000)$. Note that several relatives might live at the same street number.

For each test case, your program must write the minimal sum of distances from the optimal Vito's house to each one of his relatives. The distance between two street numbers s_i and s_j is $d_{ij} = |s_i - s_j|$.

Sample Input

```
2
2 2 4
3 2 4 6
```

Sample Output

```
2
4
```