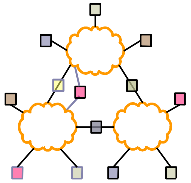
Summary of Sorting Algorithm for Assignment

Van-Hung NGUYEN

Email: nguyenvanhung.uet@gmail.com

August, 30th 2018



Outline

1) Assignment
2) Sorting Algorithm Analysis
3) Experiment and Evaluation

Outline

1) Assignment
2) Sorting Algorithm Analysis
3) Experiment and Evaluation

Assignment

- Write a sorting program that can sort 1,000,000 integers in less than 100ms
  - Input file contains K numbers. Program should work with other input files with different K and different order of integers.
  - Tasks:
    - Sort the first ‘N’ numbers in a file using a sorting algorithm of your choice
    - Measure the running time
  - A given N value, read the first N integers from the input file, put them into an array of integers, and sort them using your sorting algorithm.
    - if N > K, then program should sort all K numbers in the file correctly
  - Program must output the sorted result as well as the running time of the program in milliseconds.

Assignment

- For assignment, the following algorithms are evaluated:
  - Selection Sort
  - Insert Sort
  - Merge Sort
  - Quick Sort

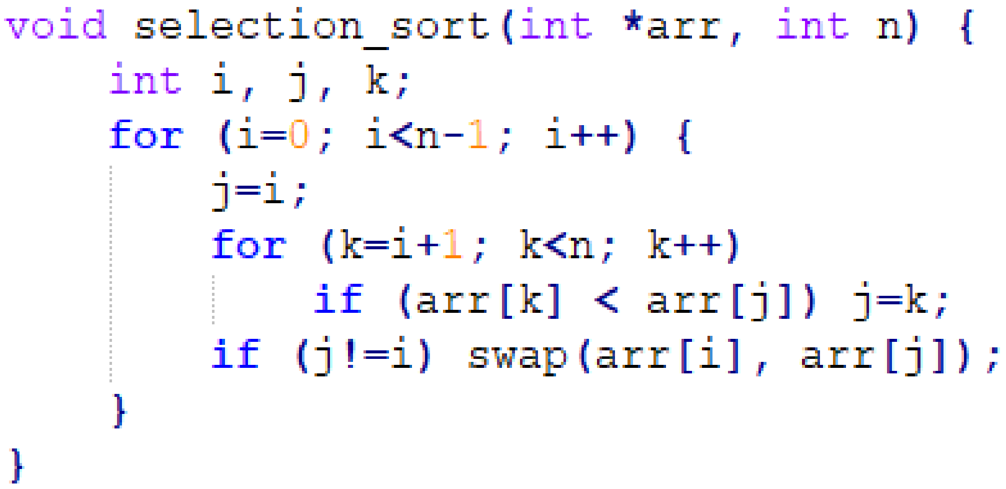
- Based on the evaluated results, the best performance algorithm will be selected.

Outline

1) Assignment
2) Sorting Algorithm Analysis
3) Experiment and Evaluation

Selection Sort

- Idea : for each i from 0 to n-2, find the smallest element in the suffix arr[i..n-1] and swap that element with arr[i]

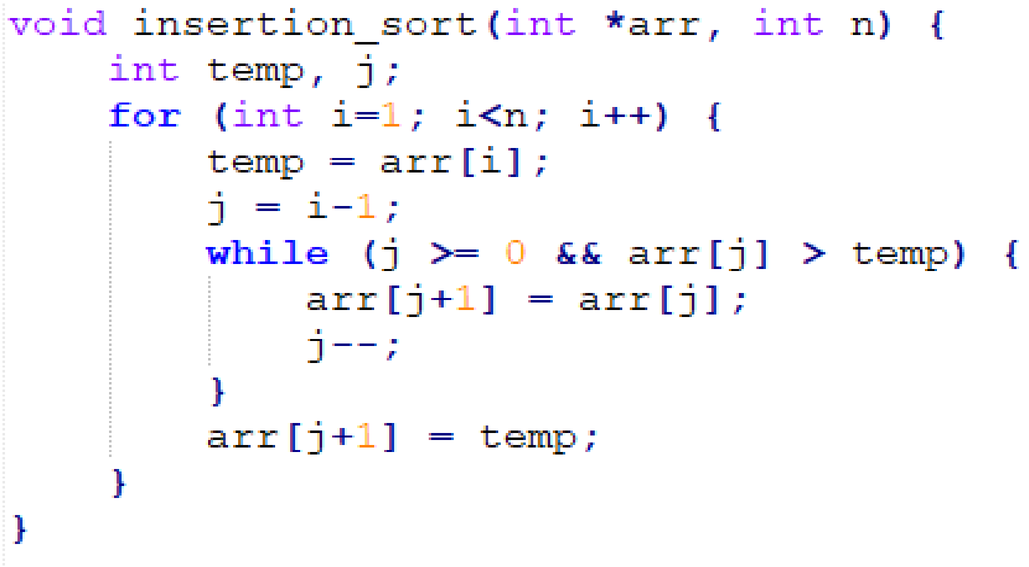


```
void selection_sort(int *arr, int n) {
    int i, j, k;
    for (i=0; i<n-1; i++) {
        j=i;
        for (k=i+1; k<n; k++)
            if (arr[k] < arr[j]) j=k;
        if (j!=i) swap(arr[i], arr[j]);
    }
}
```

- The worst case run time for Selection Sort: O(n^2)
  - Selection sort always run in time O(n^2) even when the input is already sorted

Insertion Sort

- Idea : the input is A[0 .. n - 1]. For each i from 1 to n-1, we find the right place in A[0...i-1] (which is already sorted) to insert A[i]



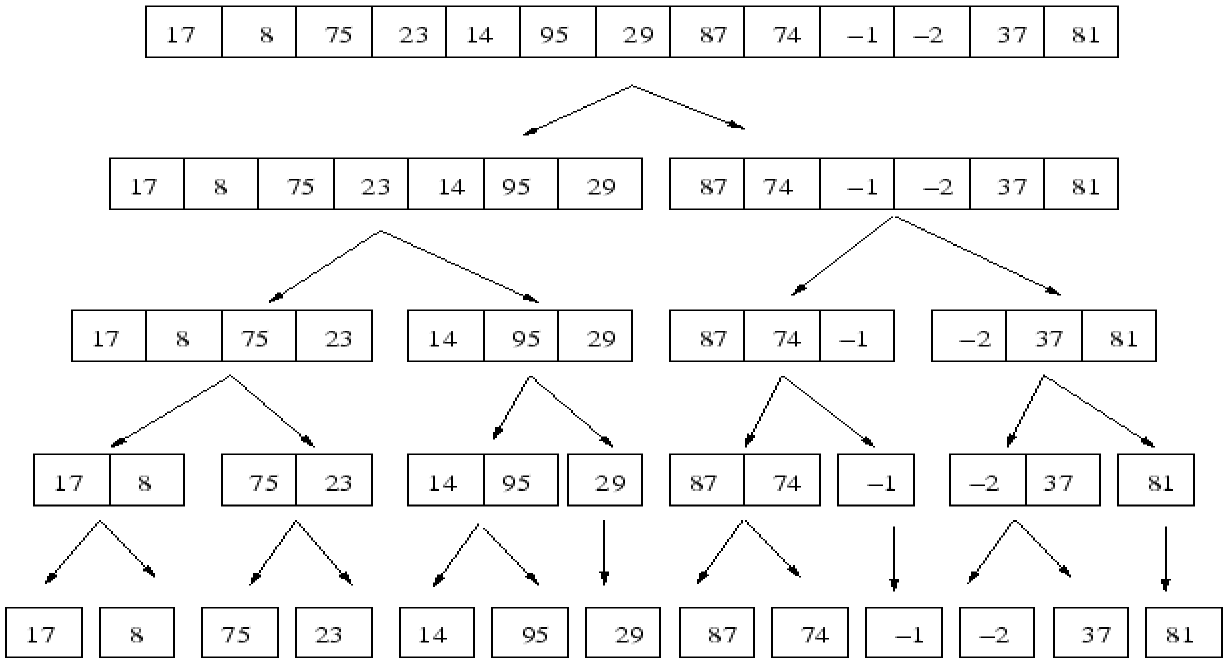
```
void insertion_sort(int *arr, int n) {
    int temp, j;
    for (int i=1; i<n; i++) {
        temp = arr[i];
        j = i-1;
        while (j >= 0 && arr[j] > temp) {
            arr[j+1] = arr[j];
            j--;
        }
        arr[j+1] = temp;
    }
}
```

- The worst case run time for Insertion Sort: O(n^2)

- It is stable sorting algorithm where the input data points have equal values maintain their relative order in the output.

Merge Sort

- Idea : splitting the input array into two halves, recursively sort the left and the right halves, and then *merge* the two halves.
- Example



17 8 75 23 14 95 29 87 74 –1 –2 37 81

17 8 75 23 14 95 29 | 87 74 –1 –2 37 81

17 8 75 23 | 14 95 29 | 87 74 –1 | –2 37 81

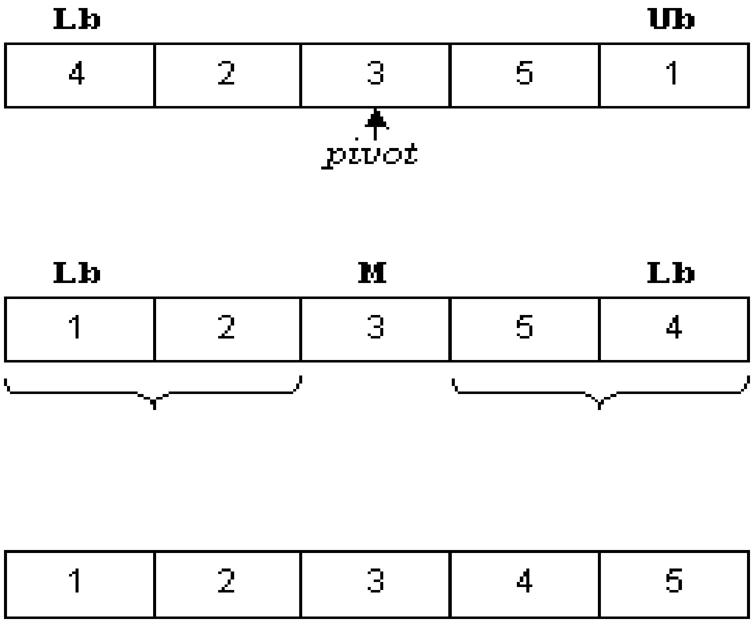
17 8 | 75 23 | 14 95 | 29 | 87 74 | –1 | –2 37 | 81

17 | 8 | 75 | 23 | 14 | 95 | 29 | 87 | 74 | –1 | –2 | 37 | 81

- Merge sort is a classic divide and conquer algorithm which has a run time O(*nlog(n)*)
- Even when the input is already sorted it still takes O(*nlog(n)*) time because of all the copying.

Quick Sort

- Idea has 3 steps:
  - The partition step: first, we rearrange items in A[p .. q] such that there is an index p ≤ r ≤ q where A[i] < A[r] for all i = p..r-1 and A[r] ≤ A[i] for all i = r + 1 .. q.
  - Recursively sort A[p .. r-1]
  - Recursively sort A[r+1 .. q]
- Example:



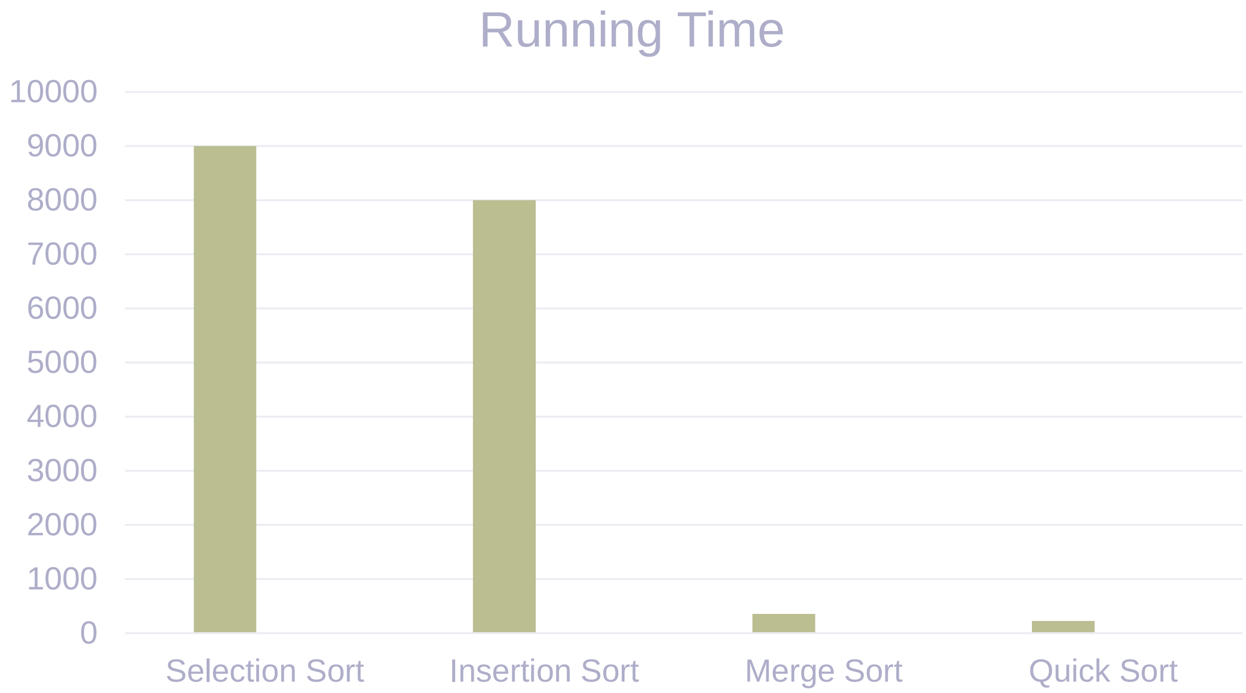
- Overall run time O(*nlog(n)*)

Outline

1) Assignment
2) Sorting Algorithm Analysis
3) Experiment and Evaluation

Experiment and Evaluation

- Input data “hw1_input.txt”.
- Due to the limitation of CPU, running time for Selection and Insertion Sort is incorrect



- With input data “hw1_input.txt”., quick sort algorithm has the best result: 229 ms.