MIDDLE EAST TECHNICAL UNIVERSITY

CENG 315
HW #2

Onur Yılmaz

Analysis of Running Times

Running times are recorded as shown in the table below. Plot of these times is provided in the next page.

	Running Times (milliseconds)			
Input Size	**Counting Sort**	**Shell Sort**	**Quick Sort**	**Radix Sort**
50	0,116	0,088	0,012	0,063
100	0,179	0,138	0,025	0,09
1.000	0,727	0,431	0,272	0,361
5.000	0,862	3,313	1,482	1,955
10.000	1,628	6,287	3,588	4,529

These times are recorded while sorting randomly generated numbers. As input size increases, fastest to slowest algorithms are as following: Counting Sort, Quick Sort, Radix Sort, Shell Sort.

Explanation of Algorithm for Mine Workers

Since greedy algorithm cannot achieve the minimum time for this problem, instead of two, in each step three next workers are considered. For this reasoning, three main cases are considered:

If there is no torch at down,

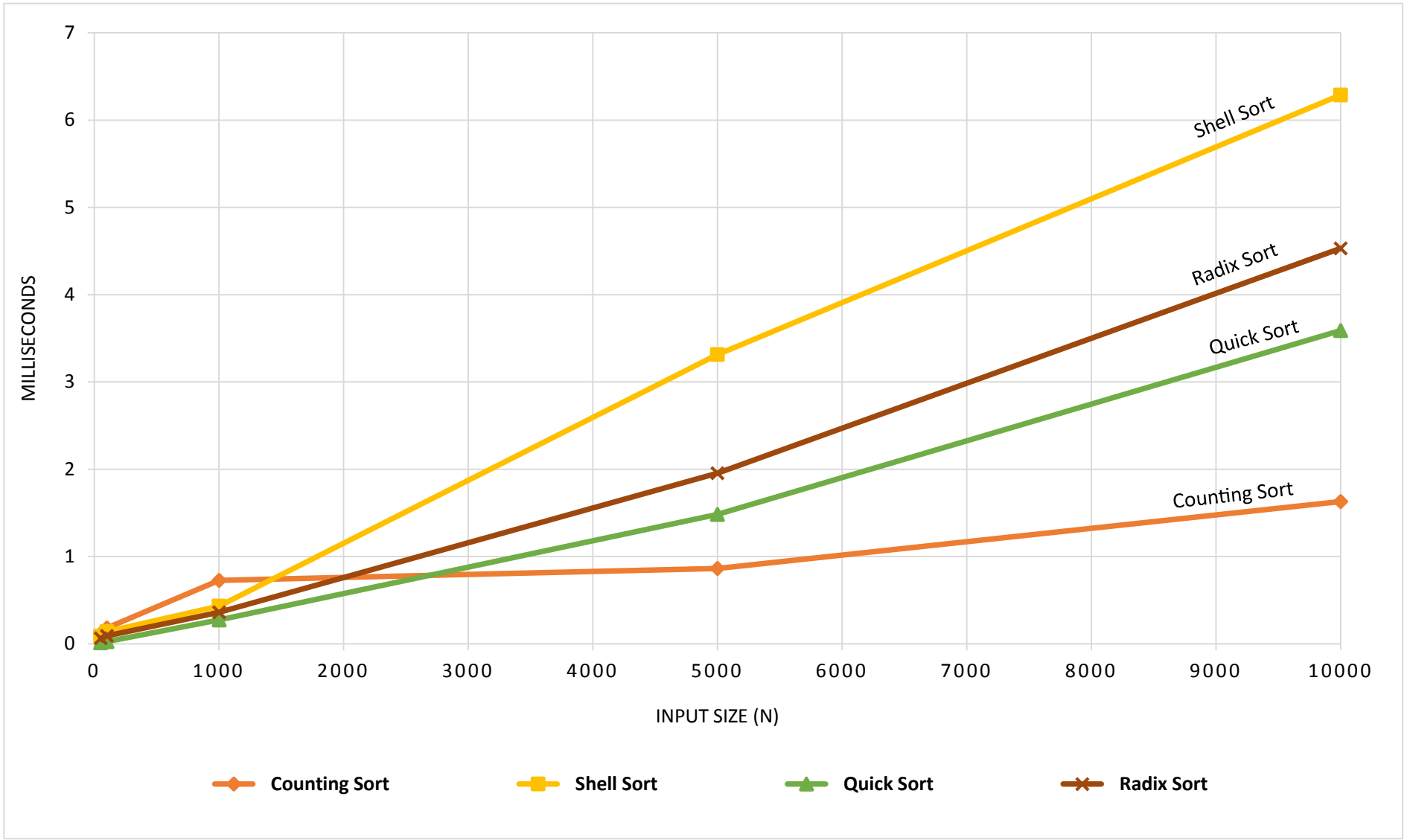
Send the fastest worker at the upside with the torch to downside.

If the two quickest workers are at downside,

Send both of them to upside.

Otherwise,

Send the two slowest workers to upside.


7
6
5
4
3
2
1
0
MILLISECONDS
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
INPUT SIZE (N)
Shell Sort
Radix Sort
Quick Sort
Counting Sort
Counting Sort
Shell Sort
Quick Sort
Radix Sort

Explanation of Sorting Algorithm

For the fourth sorting algorithm, I chose the Radix Sort Algorithm. The reason of choosing this method was to learn the implementation of Radix Sort in C++.

Radix sort is based on comparing each digit in itself by using a stable sorting algorithm. Starting from the least significant digit to most significant digit, it sorts elements in discrete steps. Number of these steps is limited by the characteristics of the digits and input. For instance, in integers, logarithmic base of 10 is considered, so the maximum number of passes is limited by the $\log_{10}$ [max (input)]. For integers, in each pass, all numbers are sorted according to their interested digit.

Pseudo code of this algorithm (using counting sort as a stable sorting) is as following:

```
RadixSort(Input[],Sorted[],NumberOfElements)

Exponent = 1

While max(Input[]) / Exponent > 0 do

      // CountingSort(Input[], Counts[], NumberOfElements)

               for i = 1 to NumberOfElements do
                  Counts[i] = 0

               for j = 1 to NumberOfElements) do
                  Counts[Input[i] / Exponent % 10]++

               for i = 1 to 10 do
                  Counts[i] = Counts[i] + C[i-1]

               for j = 1 to NumberOfElements do
                  Sorted[Counts[[Input[i] / Exponent % 10]]] = Input[j]
                  Counts[[Input[i] / Exponent % 10]]--

Exponent = Exponent * 10
```