CS469 Assignment 2

Tianye Zhao, Wenda Yang

March 7, 2019

Exercise 1

Given an integer array $Array$
$pointer1 \leftarrow Array$
$pointer2 \leftarrow Array + Array.len()$
while $pointer1$ **not** $= pointer2 + 1$ **and** $pointer1$ **not** $= pointer2$ **do**
  $swap(*pointer1, *pointer2)$
  $pointer1 \leftarrow pointer1 + 1$
  $pointer2 \leftarrow pointer2 - 1$
end while

Exercise 2

Given an integer array $Array$
Create an integer array $valueArray$ holds values in $Array$
Create an integer array $counterArray$ holds counters for respective values in $valueArray$
$pointer \leftarrow Array$
$pointer1 \leftarrow valueArray$
$pointer2 \leftarrow counterArray$
while $pointer$ **not** $= Array + Array.len() + 1$ **do**
  if $*pointer$ **not** in $valueArray$ **then**
    $*pointer1 \leftarrow *pointer$
    $*pointer2 \leftarrow 1$
    $pointer1 \leftarrow pointer1 + 1$
    $pointer2 \leftarrow pointer2 + 1$
  else
    $idx \leftarrow$ index of $*pointer$ in $valueArray$
    $counterArray[idx] \leftarrow counterArray[idx] + 1$
    if $counterArray[idx] > 1$ **then**
      print "There are one or more duplicates."
      return
    end if
  end if
  $pointer \leftarrow pointer + 1$
end while
print "There are no duplicates."
return

Exercise 3

Deletion in dynamic array:

Given an integer array $Array$
$idx \leftarrow$ index of the value to delete
$pointer \leftarrow Array + idx$
while $pointer$ **not** $= Array + Array.len()$ **do**
$*pointer \leftarrow *(pointer + 1)$
$pointer \leftarrow pointer + 1$
end while
Delete $*pointer$
$Array.len \leftarrow Array.len - 1$

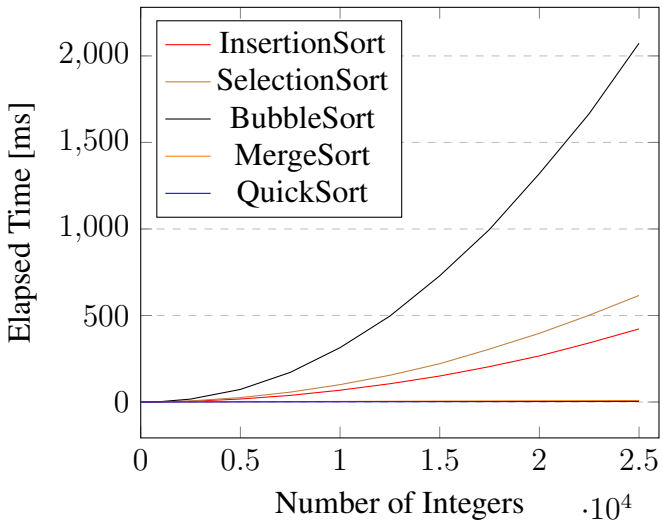
Insertion in dynamic array:

Given an integer array $Array$
$value \leftarrow$ value to insert
if $Array.isFull$ **then**
Create $newArray[2 * Array.size]$
Copy values in $Array$ to $newArray$
Free memory of $Array$
$Array \leftarrow newArray$
end if
Append $value$ to $Array$
$Array.len \leftarrow Array.len + 1$

Exercise 4

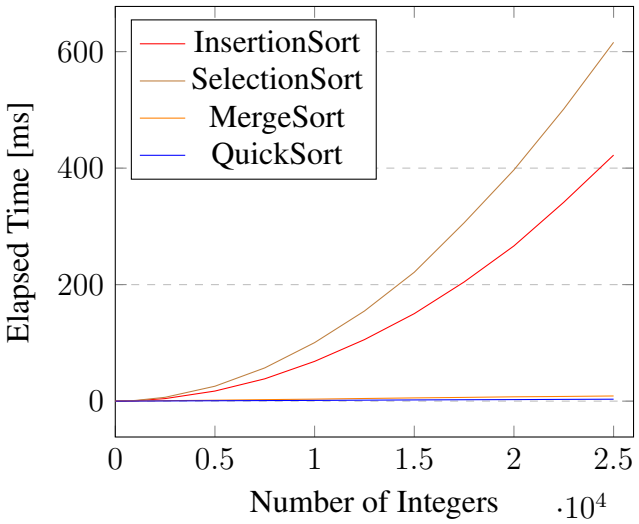
Random Array

Elapsed time of different sorting algorithms for random array



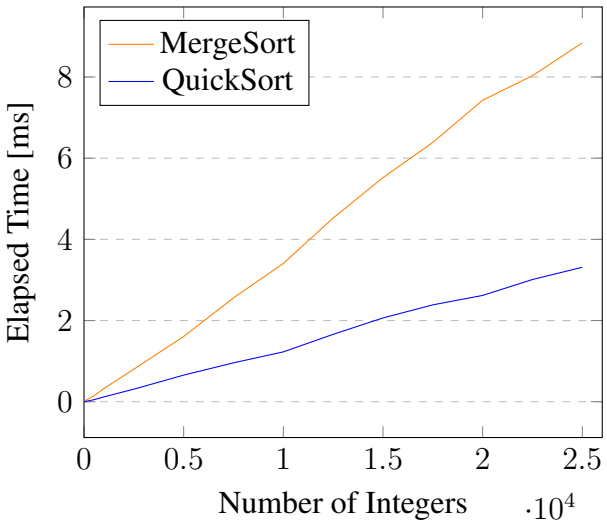
From above, we know Bubble Sort is the slowest sorting algorithm for large random arrays. For n elements, in round i from 0 to $n - 1$, we would perform $n - i - 1$ comparisons and estimated $(n - i - 1)/2$ swaps.

Elapsed time for random array (Exclude BubbleSort)



Here we can see Selection Sort performs better than Insertion Sort for large random arrays. In Selection Sort, for n elements, in round i from 0 to $n-1$, we would perform $n-i-1$ comparisons and only one swap comparing to Bubble Sort. In Insertion Sort, for n elements, in round i from 0 to $n-1$, we would perform approximate $i/2$ comparisons and only 1 swaps.

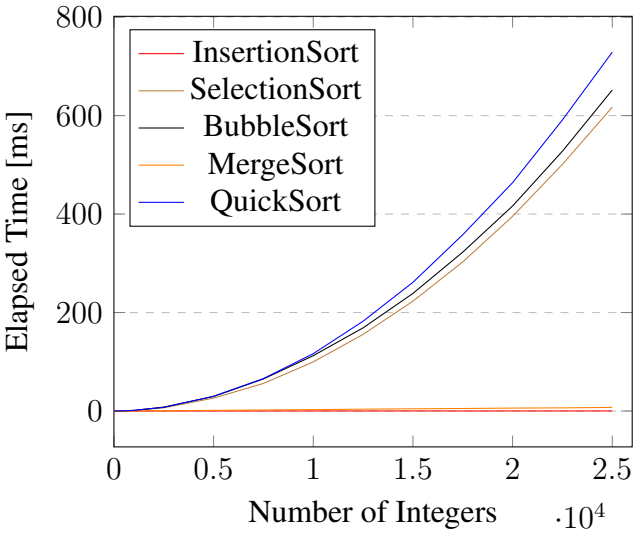
Elapsed time for random array (MergeSort & QuickSort)



Obviously, Merge Sort and Quick Sort are the best two sorting algorithm for large random arrays. For Quick Sort, the pivot can be reduced as it is already sorted in each recursion, whereas the array is strictly split to half for Merge Sort in each recursion. Thus Quick Sort performs better than Merge Sort.

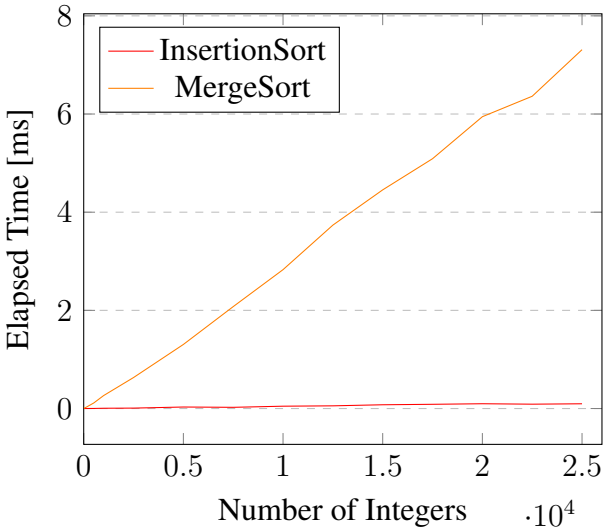
Ordered Array

Elapsed time of different sorting algorithms for ordered array



From above, we know Quick Sort becomes one of the worst sorting algorithm for ordered array. Because it can only place one element which is the pivot value in right position each recursion. Selection Sort costs basically as long as sorting random array, as it would still carry the same operation (select the minimum element and place in respective position each round) in all case scenario.

Elapsed time for ordered array (InsertionSort & MergeSort)



As we can see from here, there is also no big sorting time difference between these two kinds of arrays for Merge Sort.

Elapsed time of InsertionSort for ordered array

Insertion Sort is the best algorithm for ordered array, as it performs strictly one comparison each round.