Homework 4

Rumen Mitov

March 16, 2024

Problem 6.1

Algorithm 1 BubbleSort(A, lo, hi)

```
for i ← lo; i < hi; i ← i + 1 do                ▷ Loop through all elements
    for j ← i; j < hi − 1; j ← j + 1 do
        if A[j] > A[j + 1] then                 ▷ Check if adjacent element needs to be
swapped with current
            Swap(A[j], A[j + 1]);
        end if
    end for
end for
```

Bubble sort has to iterate over the array an amount of times as the size of the array, regardless of how the elements in the array are arranged. Hence:

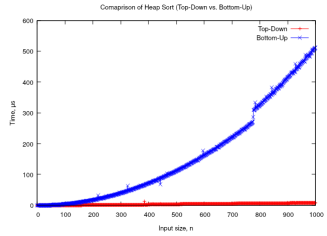
$T(n) = \Sigma_{i=0}^{n-1} n - i$

$T(n) = \frac{n(n+1)}{2}$

$T(n) = \Theta(n^2)$

Insertion and bubble sort are stable because they only perform a swap on two elements if they do not meet the comparison condition. Since equality is assumed to meet the comparison condition, elements are not swapped, hence algorithms are stable.
Merge sort is also stable because during the merge step, if the top elements of both arrays are the same, the left element should be handled first, thus maintaining stability.
Heap sort is stable, because the structure of the array is preserved (we only extract the top element and then heapify-down).

Insertion and heap sort are the only two algorithms that are adaptive. If part of the array is already sorted, insertion sort does not need to check it, hence it speeds the process up. In the case of heap sort, the heap data structure ensures that the data is weakly sorted and that existing order is what makes heap sort so performant.
Merge and bubble sort are not adaptive because they will go through all the iterations, regardless of an existing order in the input.



When comparing the bottom-up apprach to top-down heap-sort, we can clearly see that bottom-up will be less efficient. This is because we are performing an extra 2 swaps per iteration in the heap sort. And the assumption that these swaps are rare cannot be justified as when we swap the new root with a leaf, we are breaking the heap property.