Foundations of Computer Science (Semester 2) – 2015

Assessed Exercise Sheet 6 – 10% of Continuous Assessment Mark

Deadline : 11pm Sunday 1st March, via Canvas

Question 1 (16 marks)

Shaker Sort (also called Bidirectional Bubble Sort) successively compares adjacent pairs of items in an array, and exchanges them if they are in the wrong order. It alternatively passes forwards and backwards through the array until no more exchanges are needed. Explain why this algorithm is guaranteed to terminate with the array sorted.

There are a finite number of permutations of a finite array, so there must be a finite number of pairs of items in the wrong order. Swaps can only decrease the number of pairs of items in the wrong order. Each pass will swap at least one pair, and decrease the number of items in the wrong order. Thus after a finite number of passes the number of items in the wrong order must reach zero, at which point the algorithm terminates with the array sorted.

Work through the sorting of array [3, 2, 4, 5, 1, 6] using this algorithm, writing down the direction of the pass at each swap, the swaps that occur, and the array after each swap.

Start		[3, 2, 4, 5, 1, 6]
Forward	3 and 2 swap	[2, 3, 4, 5, 1, 6]
Forward	5 and 1 swap	[2, 3, 4, 1, 5, 6]
Backward	1 and 4 swap	[2, 3, 1, 4, 5, 6]
Backward	1 and 3 swap	[2, 1, 3, 4, 5, 6]
Backward	1 and 2 swap	[1, 2, 3, 4, 5, 6]
Forward	no swaps	[1, 2, 3, 4, 5, 6]

Question 2 (16 marks)

The following code will sort an array `a` of finite size `n`:

```
for( i = 1 ; i < n ; i++ ) {
    j = i;
    c = a[j];
    while( j > 0 && c < a[j-1] ) {
        a[j] = a[j-1];
        j--;
    }
    a[j] = c;
}
```

Which sorting algorithm discussed in the lectures is this?

It is Insertion Sort.

Explain why it is guaranteed to terminate with the array `a` sorted.

It is guaranteed to terminate because the `for` loop executes exactly once for each value of `i` from 1 up to finite `n-1`, and the while loop executes at most once for each value of `j` from finite `i` down to 1. The final array a will be sorted because each of the `n-1` iterations through the `for` loop brings one more item forward to its correct position in the array.

Work through the sorting of array [5, 3, 7, 8, 1, 9] using this algorithm, writing down the values of `i, j, c` and the array `a` at the end of each iteration of the `for` loop.

i	j	c	a
1	0	3	[3, 5, 7, 8, 1, 9]
2	2	7	[3, 5, 7, 8, 1, 9]
3	3	8	[3, 5, 7, 8, 1, 9]
4	0	1	[1, 3, 5, 7, 8, 9]
5	5	9	[1, 3, 5, 7, 8, 9]

Question 3 (16 marks)

The following code will also sort an array `a` of finite size `n`:

```
for( i = 0 ; i < n-1 ; i++ ) {
    k = i;
    for( j = i+1 ; j < n ; j++ ) {
        if( a[j] < a[k] ) k = j;
    }
    c = a[i];
    a[i] = a[k];
    a[k] = c;
}
```

Which sorting algorithm discussed in the lectures is this?

It is Selection Sort.

Explain why it is guaranteed to terminate with the array `a` sorted.

It is guaranteed to terminate because the outer `for` loop executes exactly once for each value of `i` from 0 up to finite `n-2`, and the inner for loop executes at most once for each value of `j` from finite `i+1` up to finite `n-1`. The final array `a` will be sorted because for each of the `n-1` iterations through the outer `for` loop, the inner `for` loop determines the smallest element k in the unsorted end of the array, and that is then swapped with the first item of the unsorted part. Thus the items are swapped into the correct sorted position one at a time starting from the smallest, ending up with a fully sorted array.

Work through the sorting of array [5, 3, 7, 8, 1, 9] using this algorithm, writing down the values of `i, j, k` and the array `a` at the end of each iteration of the `i for` loop.

i	j	k	a
0	6	4	[1, 3, 7, 8, 5, 9]

1	6	1	[1, 3, 7, 8, 5, 9]
2	6	4	[1, 3, 5, 8, 7, 9]
3	6	4	[1, 3, 5, 7, 8, 9]
4	6	4	[1, 3, 5, 7, 8, 9]

Question 4 (18 marks)

One often needs to sort items which might have identical keys (e.g., ages in years) in a way that keeps items with identical keys in their original order (e.g., alphabetical). So, if we denote the original order of an array of items by subscripts, we want the subscripts to end up in order for each set of items with identical keys. For example, if we start out with the array $[5_1, 4_2, 6_3, 5_4, 6_5, 7_6, 5_7, 2_8, 9_9]$ it should be sorted to $[2_8, 4_2, 5_1, 5_4, 5_7, 6_3, 6_5, 7_6, 9_9]$ and not to $[2_8, 4_2, 5_4, 5_1, 5_7, 6_3, 6_5, 7_6, 9_9]$. Sorting algorithms which do this are said to be *stable*. For each of Bubble Sort, Insertion Sort and Selection Sort, say whether that algorithm is stable, and explain why. For cases where the algorithm is not stable, give a simple example that illustrates why.

Bubble Sort – This is stable because no item is swapped past another unless they are in the wrong order. So items with identical keys will have their original order preserved.

Insertion Sort – This is stable because no item is swapped past another unless it has a smaller key. So items with identical keys will have their original order preserved.

Selection Sort – This is not stable, because there is nothing to stop an item being swapped past another item that has an identical key. For example, the array $[2_1, 2_2, 1_3]$ would be sorted to $[1_3, 2_2, 2_1]$ which has items 2_2 and 2_1 in the wrong order.

Question 5 (18 marks)

In the lectures notes (section 6.4), a procedure `insert(v,bst)` is presented that inserts a value `v` into a binary search tree `bst`. Summarize, using a list of possible cases that need to be dealt with, how that procedure works.

There are four basic cases `insert(v,bst)` has to deal with:

1. `bst` is empty: so create a new tree containing only `v` as root
2. `v < root(bst)`: so call `insert(v,left(bst))`
3. `v > root(bst)` : so call `insert(v,right(bst))`
4. `v == root(bst)` : meaning `v` is a duplicate, so return error

Describe the simplest possible ways the procedure could be modified to accommodate duplicate entries.

To accommodate duplicate entries, case 4 needs changing. The obvious way to proceed is to delete case 4 by combining it with either case 2 (by changing < to <=) or case 3 (by changing > to >=).

Comment on how the possible modifications would affect the stability of a Treesort algorithm

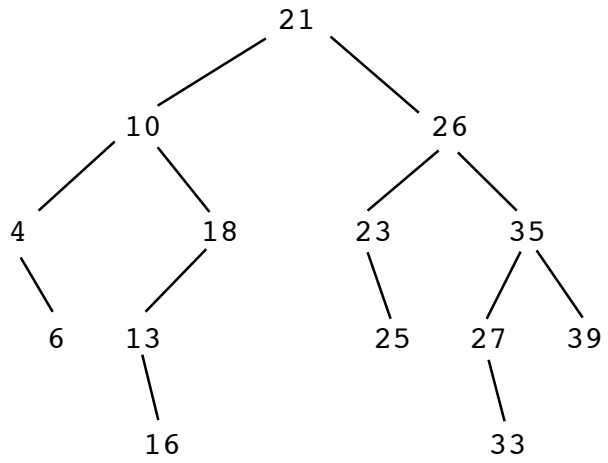
based on it.

Merging case 4 with case 2 would lead to an unstable Treesort algorithm, because a later duplicate would always be added to the left of the existing item in the tree, and thus end up before it in the sorted array. Merging case 4 with case 3 would lead to a stable Treesort algorithm, because a later duplicate would always be added to the right of the existing item in the tree, and thus remain after it in the sorted array.

Question 6 (16 marks)

Show how the array [21, 26, 23, 10, 18, 35, 13, 4, 16, 6, 25, 27, 33, 39] would be sorted if the Treesort algorithm was used. Make it clear what is being done at each stage, and show the data structure at the key stages.

The first stage is to insert the array into a binary search tree, which results in:



Then recursively extract the items from the tree in order left, root, right to give the sorted array [4, 6, 10, 13, 16, 18, 21, 23, 25, 26, 27, 33, 35, 39].