Problem Set 10

Justin Ely

615.202.81.FA15 Data Structures

10 November, 2015

---

1)

1a)

For a simple insertion sort and a sorted input file, n comparisons and 0 interchanges are performed.

For a file in reverse order, the number of comparisons on each pass will increase, as the subsequent value will need to be checked against an ever increasing number of already sorted elements. Last will need n, then (n-1), n-2, n-3, etc, etc. This averages out to $\frac{n^2}{2}$. Each pass still only does a single exchange, so n exchanges are performed.

For the alternating file, as the lower half is sorted, the top half becomes sorted as well. This will lead to n/2 exchanges, and at max n/2 comparisons for the second to largest element: $\frac{n^2}{4}$ comparisons.

2)

Insertion sort is the most efficient algorithm. Selection sort is the least-efficient, always requiring n^2 operations and not being able to take advantage of early-out conditions. Insertion and Bubble are both $O(n^2)$ and best-case ($O(n)$, but Bubble sort takes $\sim \frac{n^2}{2}$ comparisons while insertion sort takes $\sim \frac{n^2}{4}$.

3)

a)

This scenario requires (n*m) - 1 comparisons: the smallest value will alternate between a and b until only the last value remains.

b)

This scenario requires only n comparisons. After each element in a is compared to the first value in b, (taking n comparisons), all values of b can simply be inserted without comparison since they are already assumed to be in order.

4)

a)

For this scenario, n/2 + m comparisons will need to be made. The first half of a and all of b will be checked, after which b will be empty and the only remaining items in a can simply be inserted.

b)

Only 1 comparison will be performed. The only element in b will be compared against a and inserted, after which all elements in a can be inserted without comparisons.

c)

Only n comparisons will need to be made. Each element of a will be compared to b and inserted; the only element in b will be inserted after with no comparisons.

5)

For an unordered table, only a sequential search can be employed. In this case, if the key is not in the table, the search will need to go through every single record and need to do n comparisons.

For an ordered set of data, the order can be exploited to only search with more efficient strategies. Binary search can be used with for $log(n)$ performance. Interpolation search can be used in this scenario, but only if the values are evenly spaced.

6)

For an unordered table, only a sequential search can be employed. In this case, the required number of operations will average out to $\frac{n}{2}$.

For ordered data, binary search will find the value in $log(n)$. Interpolation search will find the value in $log(log(n))$, if the data is evenly spaced, otherwise it will approach $O(n)$.

7)

As before, for an unordered table, only a sequential search can be employed. In this case, the required number of operations will still average out to $\frac{n}{2}$, provided the number of repeat keys is small compared to n.

For ordered data, binary search will find the value in $log(n)$. Interpolation search will find the value in $log(log(n))$, if the data is evenly spaced, otherwise it will approach $O(n)$.

8)

As before, for an unordered table, only a sequential search can be employed. In this case, the required number of operations will always be n, as you cannot know that a repeat value is remaining in the rest of the set unless each value is checked.

For ordered data, after the first value is found then the repeat values can be found trivially. Finding the first value will be the same order as the previous problem.