Big O Sorting & Searching Summary

Sorts

1) **Selection**
a) Big O best case time - $O(n^2)$
b) Big O ave case time - $O(n^2)$
c) Big O worst case time - $O(n^2)$
d) worst case data set - reverse order
e) space requirements - one array
f) other characteristics or comments - two nested loops, terribly, slow sort, inefficient for n>100 data sets, many comparisons, could be many assignments if in reversed order, only one swap per pass of the inner loop, no early exit

2) **Bubble**
a) Big O best case time - $O(n)$
b) Big O ave case time - $O(n^2)$
c) Big O worst case time - $O(n^2)$
d) worst case data set - reverse order since a lot of values must "bubble down"
e) space requirements - one array
f) other characteristics or comments – two nested loops (for nested in a while typically) with early exit in the outer loop, good when the data is mostly sorted to start with, could cause a lot of swaps

3) **Insertion**
a) Big O best case time - $O(n)$
b) Big O ave case time - $O(n^2)$
c) Big O worst case time - $O(n^2)$
d) worst case data set - reverse order
e) space requirements - only one array if you are efficient but commonly done with two arrays (1 unsorted, other sorted), two nested loops
f) other characteristics or comments – good when the data is mostly sorted to start with, absolutely terrible in many cases, aka poker hand sort, early exit from inner loop

5) **Quick**
a) Big O best case time - $O(n \log n)$
b) Big O ave case time - $O(n \log n)$
c) Big O worst case time - $O(n^2)$
d) worst case data set - when poor pivot values are chosen such as if data is in order and first element of array is used as pivot element
e) space requirements - Big O for space is $O(n)$ in worst case, one array in each recursive stack frame
f) other characteristics or comments – on average this is the fastest known algorithm, may be worthwhile to randomize the data so that data is not close to sorted before using quick sort

6) **Merge**
a) Big O best case time - $O(n \log n)$
b) Big O ave case time - $O(n \log n)$
c) Big O worst case time - $O(n \log n)$
d) worst case data set - none
e) space requirements - high, log n recursive stack frames each with a separate array
f) other characteristics or comments - good to use if data compared to the quick sort if the data is almost sorted

7) **Heap**
a) Big O best case time - $O(n \log n)$
b) Big O ave case time - $O(n \log n)$
c) Big O worst case time - $O(n \log n)$
d) worst case data set - more swaps if data is in reverse order, but no real worst case since the order of the input items do not significantly affect its efficiency
e) space requirements - one array though in some implementations people use two arrays (one for the

heap and one for the sorted values), considered to be an "in-place" sort since it requires no temporary storage

f) other characteristics or comments - very efficient for large n, two phases to th is algorithm: putting values into a heap (i.e. binary tree with the heap property) and then "picking" the sorted values out of the heap one-by-one

8) **Radix**
a) Big O best case time - O(n)
b) Big O ave case time - O(n)
c) Big O worst case time - O(n)
d) worst case data set - if data elements are large it will consume a lot of memory
e) space requirements - excessive
f) other characteristics or comments - not suitable for all types of data such as floating-point values, has a large constant of proportionality despite being O(n)

Searches

1) **Sequential**
a) Big O best case time - O(1)
b) Big O ave case time - O(n)
c) Big O worst case time - O(n)
d) worst case data set - if key is not found
e) space requirements - one array
f) other characteristics or comments - easy to write code, okay for small n

2) **Binary**
a) Big O best case time - O(1)
b) Big O ave case time - O(log n)
c) Big O worst case time - O(log n)
d) worst case data set - if key is not found
e) space requirements - one array
f) other characteristics or comments - data must be ordered

3) **Lookup & Hash Tables**
a) Big O best case time - O(1)
b) Big O ave case time - O(1)
c) Big O worst case time - O(n)
d) worst case data set - if there are many elements that hash to the same bin (i.e. many collisions)
e) space requirements - excessive
f) other characteristics or comments - depends on the hash function and whether duplicate elements.
If there are duplicate elements, the efficiency of the collision strategy is important.