Practice Problems

1) **When will the Quicksort perform poorly (worst case)?**

2) **We have a system running insertion sort and we find that it's completing faster than expected. What could we conclude about the input to the sorting algorithm?**

3) **Explain which sorting algorithm you would use to sort the input array the fastest and why you chose this sorting algorithm. An array of n Comparable objects that is sorted except for k randomly located elements that are out of place (that is, the list without these k elements would be completely sorted)**

4) **Which sorting algorithm will have the best time complexity for sorting this array? Select one of them: Bubble | Selection | Insertion**

   a. **1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | -3 | -4**

   b. **1 | 15 | 3 | 4 | 5 | 8 | 10 | 12 | 13 | 2**

   c. **32 | 27 | 25 | 5 | 9 | 10 | 15 | 18 | 22 | 24**

5)

Given an array, apply **Shell Sort** on it:

9	8	3	7	5	6	4	1

```
void shellSort(int array[], int n)
{
        for (int interval = n / 2; interval > 0; interval /= 2)
      {
              for (int i = interval; i < n; i += 1)
            {
                    int temp = array[i];
                    int j;
                    for (j = i; j >= interval && array[j - interval] > temp; j -=interval)
                  {
                          array[j] = array[j - interval];
                    }
                    array[j] = temp;
              }
      }
}
```

1. Perform Dry run of the code and show array at each step. [10 marks]
2. Indicate the best and worst case of given algorithm in terms of Big -Oh [5 marks]

Solutions

1) **Quicksort has a worst case runtime of Θ(N2), if the array is partitioned very unevenly at each iteration.**

2) **Input already sorted.**

3) **Sort: Insertion sort**
   Runtime: O(nk)
   Explanation: For the n – k sorted elements, insertion sort only needs 1 comparison to check that it is in the correct location (larger than the last element in the sorted section). The re-maining k out-of-place elements could be located anywhere in the sorted section. In the worst case, they would be inserted at the beginning of the sorted section, which means there are O(n) comparisons in the worst-case for these k elements. This leads to an overall runtime of O(nk + n), which simplifies to O(nk).

4)
   a. **Insertion sort**
   b. **–**
   c. **--**