# Unit 7 In-Class Assignment: Search and Sort

In this assignment, we will further explore the Search and Sort concepts in Chapter 10. Please refer to notes and textbook for definitions and details.

1. What are the similarities and differences between an Unordered Linear Search and Ordered Linear Search?

**Unordered Linear Search**: In order to determine than an object does exist in the collection, one needs to search through the entire collection.

**Ordered Linear Search**: given array is sorted, one need not necessarily search through the entire list to find a particular object or determine that it does not exist in the collection. While scanning the array from left to right, a search can now be terminated early if and when it is determined that the number being sought is less than the element currently being examined.

1. Note the similarities and/or differences between Linear and Chunk Search.

**Chunk Search:** The collection is divided into “chunks” or subsets, C, which is equal to sqrt(i), where I is the number of elements in the original set. The search key is compared to the maximum value of the first chunk. If the search key is greater than the maximum of the first chunk, the second chunk will be evaluated and so on.

**Linear Search:** Each element is compared in an unordered linear search one by one. In an ordered linear search the elements are compared and the search is terminated as soon as the key is found.

1. What is the purpose of having worst case analysis when analyzing algorithm?

Worse case analysis is helpful because it helps determine the slowest possible execution time for an algorithm and help you model how the maximum time it takes to execute the algorithm that way we will always know that the algorithm cannot be slower than the derived worst case bound. However in practice algorithms are typically much faster than their worst case bound.

1. When searching through 2000 elements in a vector, assuming that the system takes .5 nano-second to search through each element. How long will it take to search the entire vector?

0.5 nanoseconds \* 2000 elements = 1000 nanoseconds = 0.000001 seconds

1. If an application uses Chunk Search to find matches in 100 elements in an array. What is the ideal chunk size?

N = 100

Sqrt(n) = C

Sqrt(100) = C

10 = C

1. When searching in an array that contains 8400 elements,
2. How many comparisons must be performed in Binary Search to find a result?

T(n) = = 13 comparisons

1. How many comparisons need to be performed using Chunk Search?

T(n) = 2 \* sqrt(n) = 2 \* sqrt(8400) = 2 \* 91 = 182 comparisons

1. How many comparisons need to be performed using Linear Search?

8400 comparisons

T(n) = 8400

1. When searching in an array that contains 9750 elements,
2. How many comparisons must be performed in Binary Search to find a result?

T(n) = = 13 comparisons

1. How many comparisons need to be performed using Chunk Search?

T(n) = 2 \* sqrt(n) = 2 \* sqrt(9750)

1. How many comparisons need to be performed using Linear Search?

9750 comparisons

T(n) = 9750

1. Use the below data and provide explanation on how insertion sort can be used with the given data.

| i | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Value | 87 | 6 | 14 | 60 | 9 | 44 | 11 | 5 | 8 | 98 |

Sort begins by comparing the 87 to 6 element. If 6 than 87 6 is placed before 87 element.

**1st iteration**

| i | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Value | **6** | **87** | 14 | 60 | 9 | 44 | 11 | 5 | 8 | 98 |

**2nd iteration**

14 is compared to 6. If less than 6 it is placed before 6. If it is greater than 6 and less than 87 it is placed after 6 and before 87.

| i | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Value | **6** | **14** | **87** | 60 | 9 | 44 | 11 | 5 | 8 | 98 |

The process continues until array is fully sorted.

1. Given the below data, provide explanation on how selection sort can be used with the given data.

| i | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Value | 41 | 7 | 12 | 83 | 3 | 40 | 16 | 2 | 71 | 6 |

**1st iteration**

Selection sort examines list and selects the smallest element and swaps with first element.

| i | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Value | **2** | 7 | 12 | 83 | 3 | 40 | 16 | **41** | 71 | 6 |

**2nd iteration**

Selection sort starts at second element and searches right to find smallest element and swaps with second element.

| i | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Value | 2 | **3** | 12 | 83 | **7** | 40 | 16 | 41 | 71 | 6 |

The process continues until array is fully sorted.

1. Refer to the below data, provide explanation on how merge sort is applied.

| i | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Value | 30 | 5 | 19 | 66 | 4 | 10 | 28 | 6 |

Break list into two approximately equal pieces and recursively apply Merge-sort to each piece to produce sorted list.

**Divide once into two subsets:**

30, 5, 19, 66

4, 10, 28, 6

**Divide again into 4 subsets:**

30, 5

19, 66

4, 10

28, 6

**Divide again into 8 subsets.**

30 5 19 66

4 10 28 6

**Merge and order**

5, 30 19, 66

4, 10 6, 28

**Merge and order**

5, 19, 30, 66 4, 6, 10, 28

**Merge and order**

4, 5, 6, 10, 19, 28, 30, 66