Shell sort

- Worst-case time: 0(n^2)
- Average time: 0(n^2)
- Best-case time: 0(n log^2 n)

code

```
/*SHELL SORT*/
 public string[] ShellSort(string[] names)
 {
     int n = names.Length;

     for (int gap = n / 2; gap > 0; gap /= 2)
     {
         for (int i = gap; i < n; i++)
         {
             string temp = names[i];
             int j;

             for (j = i; j >= gap && string.Compare(names[j - gap], temp) > 0; j
-= gap)
             {
                 names[j] = names[j - gap];
             }

             names[j] = temp;
         }
     }
     return names;
 }
```

Explanation

Shell sort works by dividing the array into smaller arrays and sorting them using insertion sort.

First you must choose a gap sequence. You start with the biggest gap an perform an insertion sort on every subarray created by this gap. Then you will reduce the gap to the next value in the sequence and repeat the steps until the gap becomes 1. When this happens, it will perform a standard insertion sort to make sure the array is fully sorted.

In my code I used n (which is the length of the string array names) divided by 2 as the first gap. I then reduced the gap by dividing it by 2. The code then returns the sorted array.

Gnome sort

- Worst-case time: 0(n^2)
- Average time: 0(n^2)
- Best-case time: 0(n)

code

```
/*GNOME SORT*/
public string[] GnomeSort(string[] names)
{
    int len = names.Length;
    int i = 0;

    while (i < len)
    {
        if (i == 0)
        {
            i++;
        }
        if (string.Compare(names[i], names[i - 1], StringComparison.Ordinal) >=
0) // stringComparison.ordinal compares the unicode value of both strings
        {
            i++;
        }
        else
        {
            string temp = names[i];
            names[i] = names[i - 1];
            names[i - 1] = temp;
            i--;
        }
    }
    return names;
}
```

Explanation

Gnome sort works by repeatedly moving an element to its correct position in the array.

Gnome sort start at index 0. It compares the current element with the previous one (if a previous one exists). If the current element is smaller than the previous one, we swap the 2 elements and take a step backwards in the array. We repeat this process until the element is in it's correct position, then we move forwards again. We will repeat this process until the array is fully sorted.

If you take a look at my code, you can see that I use a function to compare the Unicode value of both strings. I use these values to sort the names. In that if else statement, you can also see the switch taking place. After the switch is done, I subtract 1 from my I;

Comparison

The most important difference between gnome sort and shell sort is that shell sort is quicker. If I had to put it into big O notation, gnome sort would be O(n) in the best case. Shell sort would be O(n log^2 n) which is an improvement. In terms of memory, they both use a constant amount of memory

0(1). They have no need to use any extra memory space, because they both sort the elements within an existing array.

FULL CODE USED IN THIS ASSIGNMENT

MAIN.CS

```
namespace Where_is_Wally
{
    internal class Program
    {
        static void Main(string[] args)
        {
            string[] unsortedArray = File.ReadAllLines("names.txt");
            string name = "Wally";
            Sorting sorting = new Sorting();
            string[] SelectionSortedArray = sorting.SelectionSort(unsortedArray);
            int result;
            /*SELECTION SORT*/
            try {
                result = sorting.BinarySearch(SelectionSortedArray, name);
                if (result != -1)
                {
                    Console.WriteLine($"Found {name} at {result}");
                }
                else
                {
                    Console.WriteLine($"Cannot find {name}");
                }
            }
            catch (Exception e)
            {
                Console.WriteLine(e);
            }

            /*BUBBLE SORT*/
            try
            {
                string[] BubbleSortArray = sorting.BubbleSort(unsortedArray);
                result = sorting.BinarySearch(BubbleSortArray, name);
                if (result != -1)
                {
                    Console.WriteLine($"Found {name} at {result}");
                }
                else
                {
                    Console.WriteLine($"Cannot find {name}");
                }
            }
            catch (Exception e)
            {
                Console.WriteLine(e);
            }

            /*INSERTION SORT*/

            try
            {
                string[] InsertionSortArray =
sorting.InsertionSort(unsortedArray);
                result = sorting.BinarySearch(InsertionSortArray, name);
                if (result != -1)
                {
```

```
                Console.WriteLine($"Found {name} at {result}");
            }
            else
            {
                Console.WriteLine($"Cannot find {name}");
            }
        }
        catch (Exception e)
        {
            Console.WriteLine(e);
        }


        /*SHELL SORT*/

        try
        {
            string[] ShellSortArray = sorting.ShellSort(unsortedArray);
            result = sorting.BinarySearch(ShellSortArray, name);
            if (result != -1)
            {
                Console.WriteLine($"Found {name} at {result}");
            }
            else
            {
                Console.WriteLine($"Cannot find {name}");
            }
        }
        catch (Exception e)
        {
            Console.WriteLine(e);
        }

        /*Gnome sort*/
        try
        {
            string[] GnomeSortArray = sorting.GnomeSort(unsortedArray);
            result = sorting.BinarySearch(GnomeSortArray, name);
            if (result != -1)
            {
                Console.WriteLine($"Found {name} at {result}");
            }
            else
            {
                Console.WriteLine($"Cannot find {name}");
            }
        }
        catch (Exception e)
        {
            Console.WriteLine(e);
        }
    }
  }
}
```

SORTING.CS

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Where_is_Wally
{
    public class Sorting
    {
        /*SELECTION SORT*/
        public string[] SelectionSort(string[] names)
        {
            for (int i = 0; i < names.Length - 1; i++)
            {
                int minIndex = i;
                for (int j = i + 1; j < names.Length; j++)
                {
                    if (string.Compare(names[j], names[minIndex]) < 0)
                    {
                        minIndex = j;
                    }
                }
                if (minIndex != i)
                {
                    string temp = names[i];
                    names[i] = names[minIndex];
                    names[minIndex] = temp;
                }
            }
            return names;
        }

        /*BUBBLE SORT*/
        public string[] BubbleSort(string[] names)
        {
            bool swapped;
            do
            {
                swapped = false;
                for (int i = 0; i < names.Length - 1; i++)
                {
                    if (string.Compare(names[i], names[i + 1]) > 0)
                    {
                        string temp = names[i];
                        names[i] = names[i + 1];
                        names[i + 1] = temp;
                        swapped = true;
                    }
                }
            } while (swapped);

            return names;
        }
        /*INSERTION SORT*/
        public string[] InsertionSort(string[] names)
        {
            for (int i = 1; i < names.Length; i++)
            {
                string key = names[i];
                int j = i - 1;
```

```
                while (j >= 0 && string.Compare(names[j], key) > 0)
                {
                    names[j + 1] = names[j];
                    j = j - 1;
                }

                names[j + 1] = key;
            }
            return names;
        }

        /*SHELL SORT*/
        public string[] ShellSort(string[] names)
        {
            int n = names.Length;

            for (int gap = n / 2; gap > 0; gap /= 2)
            {
                for (int i = gap; i < n; i++)
                {
                    string temp = names[i];
                    int j;

                    for (j = i; j >= gap && string.Compare(names[j - gap], temp)
> 0; j -= gap)
                    {
                        names[j] = names[j - gap];
                    }

                    names[j] = temp;
                }
            }
            return names;
        }
        /*GNOME SORT*/
        public string[] GnomeSort(string[] names)
        {
            int len = names.Length;
            int i = 0;

            while (i < len)
            {
                if (i == 0)
                {
                    i++;
                }
                if (string.Compare(names[i], names[i - 1],
StringComparison.Ordinal) >= 0) // stringComparison.ordinal compares the unicode
value of both strings
                {
                    i++;
                }
                else
                {
                    string temp = names[i];
                    names[i] = names[i - 1];
                    names[i - 1] = temp;
                    i--;
                }
            }
            return names;
        }
```

```
        /*BINARY SEARCH*/
        public int BinarySearch(string[] names, string target)
        {
            int left = 0;
            int right = names.Length - 1;

            while (left <= right)
            {
                int mid = left + (right - left) / 2;

                int compareResult = string.Compare(names[mid], target);

                if (compareResult == 0)
                {
                    return mid;
                }

                if (compareResult < 0)
                {
                    left = mid + 1;
                }
                else
                {
                    right = mid - 1;
                }
            }

            return -1;
        }
    }
}
```