

## UCS 2403 Design & Analysis of Algorithms

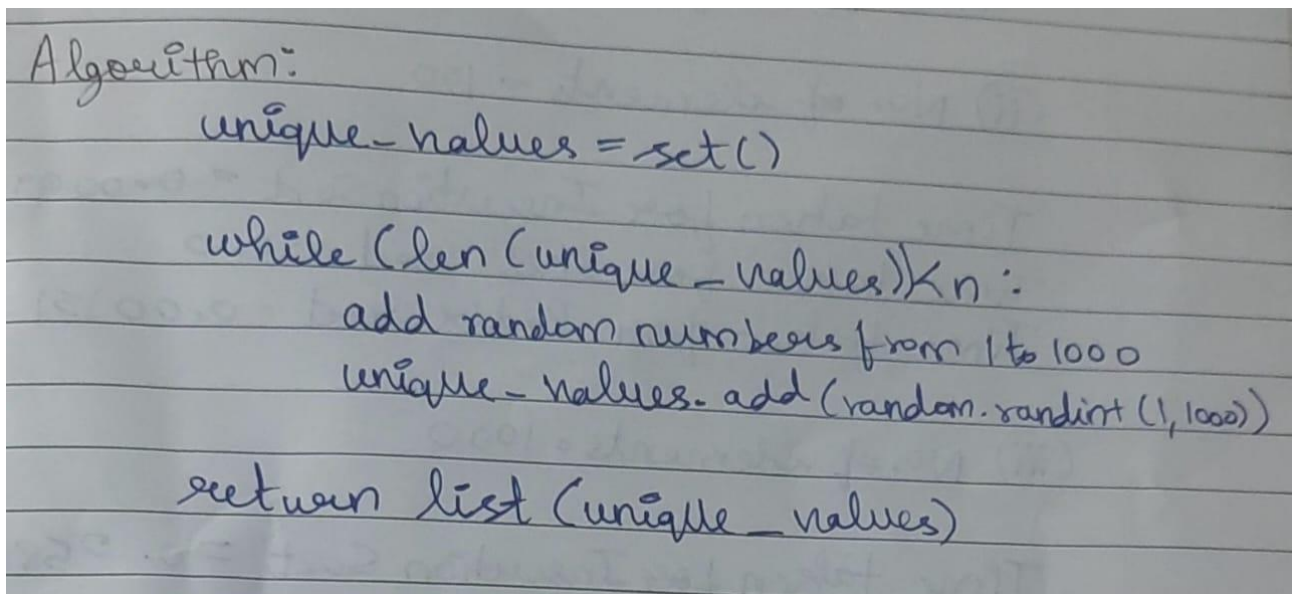
### Assignment 1

**Date of Exercise:** 22.02.2024

**Aim:** To gain understanding and proficiency in various sorting algorithms and search methods, including both recursive and non-recursive approaches

**Question 1:** Develop a Python code that takes as input a value n, and generates a list of n unique random values.

**Algorithm:**



Algorithm:

```
unique_values = set()
while (len(unique_values) < n):
    add random numbers from 1 to 1000
    unique_values.add(random.randint(1, 1000))
return list(unique_values)
```

**Code:**

```
import random
import time

def generate_unique_random_list(n):
    start = time.time()
    if n <= 0:
        return [], 0
    unique_values = set()
    while len(unique_values) < n:
        unique_values.add(random.randint(1, 1000))
    end = time.time()
    return list(unique_values), end - start

n = int(input("Enter the number of unique random values to generate: "))
random_list, time_taken = generate_unique_random_list(n)
```

```
print("Generated list of unique random values:", random_list)
print("Time taken:", time_taken, "seconds")
```

### Output:

```
PS C:\Users\Mugilkrishna D U\OneDrive\Desktop\My Files\.vscode> & "C:/Users/Mugilkrishna D U/AppData/Local/Programs/Python/Python311/python.exe" "c:/Users/Mugilkrishna D U/OneDrive/Desktop/My Files/SSN/SEM4/DESIGN AND ANALYSIS OF ALGORITHMS/LAB/1.1.py"
Enter the number of unique random values to generate: 3
Generated list of unique random values: [902, 915, 478]
Time taken: 0.0 seconds
PS C:\Users\Mugilkrishna D U\OneDrive\Desktop\My Files\.vscode> & "C:/Users/Mugilkrishna D U/AppData/Local/Programs/Python/Python311/python.exe" "c:/Users/Mugilkrishna D U/OneDrive/Desktop/My Files/SSN/SEM4/DESIGN AND ANALYSIS OF ALGORITHMS/LAB/1.1.py"
Enter the number of unique random values to generate: 5
Generated list of unique random values: [386, 994, 585, 725, 184]
Time taken: 0.0 seconds
PS C:\Users\Mugilkrishna D U\OneDrive\Desktop\My Files\.vscode> & "C:/Users/Mugilkrishna D U/AppData/Local/Programs/Python/Python311/python.exe" "c:/Users/Mugilkrishna D U/OneDrive/Desktop/My Files/SSN/SEM4/DESIGN AND ANALYSIS OF ALGORITHMS/LAB/1.1.py"
Enter the number of unique random values to generate: 10
Generated list of unique random values: [294, 166, 842, 459, 754, 242, 308, 856, 861, 734]
Time taken: 0.0 seconds
```

**Time Complexity:**  $O(n)$

**Question 2:** Develop a Python code to implement insertion sort, shell sort and radix exchange sort, and analyze their performances for arrays of the following size:

- 10
- 1000
- 1000000

**Algorithm:**

Algorithm:

(i) Insertion Sort:

```
InsertionSort(arr):  
    n = length of arr  
    if n == 1  
        return  
    for i ← 1 to n:  
        j ← i - 1  
        while j > 0 and arr[j] < arr[j + 1]:  
            swap(arr[j], arr[j + 1])  
            j ← j - 1
```

(ii) Shell Sort:

```
shellSort(arr):  
    size = len(arr)  
    gap = size // 2  
    while gap > 0:  
        for i ← gap, size:  
            key ← arr[i]  
            j ← i  
            while j > gap and arr[j - gap] > key:  
                arr[j] ← arr[j - gap]  
                j ← j - gap  
            arr[j] ← key  
            gap = gap // 2
```

(iii) Radix Sort:

```
countingSort(arr, exp):  
    n = len(arr)  
    op = [0] * n  
    count = [0] * 10  
    for i ← 0 to size n:  
        index ← (arr[i] // exp)  
        count[int(index / 10)] ← + 1  
    for i ← 1 to 10:  
        count[i] ← count[i] + count[i - 1]  
    i ← n - 1  
    while i >= 0:  
        index ← (arr[i] // exp)  
        output[count[int(index / 10)] - 1] ← arr[i]  
        count[int(index / 10)] ← - 1  
        i ← i - 1  
    for i ← 0, len(arr):  
        arr[i] ← output[i]  
radixSort(arr)  
    max1 ← max(arr)  
    exp ← 1  
    while max1 // exp > 0:  
        countingSort(arr, exp)  
        exp * = 10
```

**Code:**

```
import random
import time

n = int(input("Enter the number of elements: "))
a = random.sample(range(1, 1000000), n)
l = a.copy()

# Insertion Sort
def insertionSort(arr):
    start = time.time()
    n = len(arr)
    if n <= 1:
        return
    for i in range(1, n):
        key = arr[i]
        j = i - 1
        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
    end = time.time()
    return end - start

# Shell Sort
def shellSort(arr):
    start = time.time()
    size = len(arr)
    gap = size // 2

    while gap > 0:
        for i in range(gap, size):
            key = arr[i]
            j = i
            while j >= gap and arr[j - gap] > key:
                arr[j] = arr[j - gap]
                j -= gap
            arr[j] = key
        gap = gap // 2
    end = time.time()
    return end - start

# Counting Sort (used in Radix Sort)
def countingSort(arr, exp1):
    n = len(arr)
    output = [0] * n
```

```
count = [0] * 10
for i in range(0, n):
    index = (arr[i] // exp1)
    count[int((index) % 10)] += 1
for i in range(1, 10):
    count[i] += count[i - 1]
i = n - 1
while i >= 0:
    index = (arr[i] // exp1)
    output[count[int((index) % 10)] - 1] = arr[i]
    count[int((index) % 10)] -= 1
    i -= 1
for i in range(0, len(arr)):
    arr[i] = output[i]

# Radix Sort
def radixSort(arr):
    start = time.time()
    max1 = max(arr)
    exp = 1
    while max1 // exp > 0:
        countingSort(arr, exp)
        exp *= 10
    end = time.time()
    return end - start

t_insertion = insertionSort(a.copy())
print("Time Taken for Insertion Sort:", t_insertion)
a = l.copy()

t_shell = shellSort(a.copy())
print("Time Taken for Shell Sort:", t_shell)
a = l.copy()

t_radix = radixSort(a.copy())
print("Time Taken for Radix Sort:", t_radix)
```

**Output:**

```
PS C:\Users\Mugilkrishna D U\OneDrive\Desktop\My Files\vscode> & "C:/Users/Mugilkrishna D U/AppData/Local/Programs/Python/Python311/python.exe" "c:/Users/Mugilkrishna D U/OneDrive/Desktop/My Files/SSN/SEM4/DESIGN AND ANALYSIS OF ALGORITHMS/LAB/1.2.py"
Enter the number of elements: 10
Time Taken for Insertion Sort: 0.0
Time Taken for Shell Sort: 0.0
Time Taken for Radix Sort: 0.0
PS C:\Users\Mugilkrishna D U\OneDrive\Desktop\My Files\vscode> & "C:/Users/Mugilkrishna D U/AppData/Local/Programs/Python/Python311/python.exe" "c:/Users/Mugilkrishna D U/OneDrive/Desktop/My Files/SSN/SEM4/DESIGN AND ANALYSIS OF ALGORITHMS/LAB/1.2.py"
Enter the number of elements: 100
Time Taken for Insertion Sort: 0.0009999275207519531
Time Taken for Shell Sort: 0.0
Time Taken for Radix Sort: 0.0015103816986083984
PS C:\Users\Mugilkrishna D U\OneDrive\Desktop\My Files\vscode> & "C:/Users/Mugilkrishna D U/AppData/Local/Programs/Python/Python311/python.exe" "c:/Users/Mugilkrishna D U/OneDrive/Desktop/My Files/SSN/SEM4/DESIGN AND ANALYSIS OF ALGORITHMS/LAB/1.2.py"
Enter the number of elements: 1000
Time Taken for Insertion Sort: 0.05869865417480469
Time Taken for Shell Sort: 0.006209850311279297
Time Taken for Radix Sort: 0.01304483413696289

Enter the number of elements: 100000
Time Taken for Insertion Sort: 629.8322710990906
Time Taken for Shell Sort: 1.0704278945922852
Time Taken for Radix Sort: 0.7712833881378174
```

**Time Complexity:**

- Insertion Sort:  $O(n^2)$ ,  $\Omega(n)$ ,  $\Theta(n^2)$
- Shell Sort:  $O(n^2)$ ,  $\Omega(n \log n)$ ,  $\Theta(n \log n)$
- Radix Sort:  $O(n*d)$ ,  $\Omega(n*d)$ ,  $\Theta(n*d)$

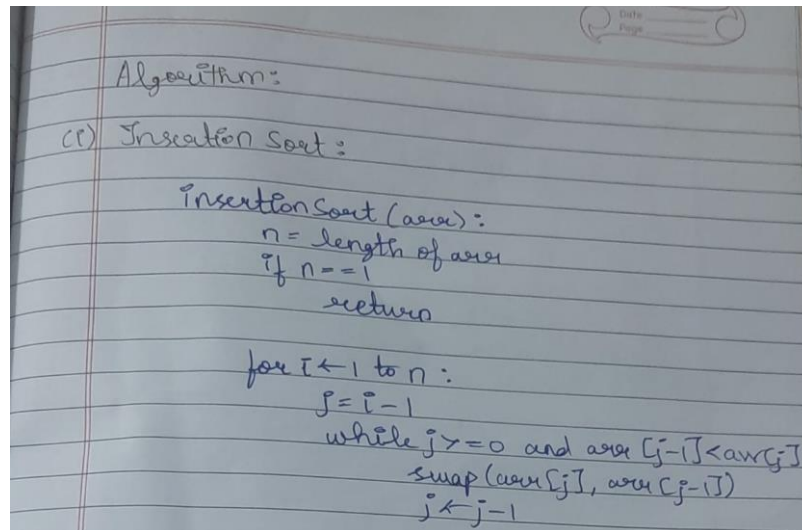
**Time Complexity Analysis table:**

No of elements	Insertion Sort (Time taken)	Shell Sort (Time taken)	Radix Sort (Time taken)
10	0.0	0.0	0.0
100	0.00099	0.0	0.00151
1000	0.05869	0.00620	0.01304
100000	629.83227	1.07042	0.77128

**Question 3:** Develop a Python code to implement insertion sort, and analyze its performance for an array of size 100000 when the input array is:

- Sorted in ascending order
- Sorted in descending order
- Not sorted

**Algorithm:**



**Code:**

```
import random
import time

n = int(input("Enter the number of elements: "))
a = random.sample(range(1, 1000000), n)
l = a.copy()

# Insertion Sort
def insertionSort(arr):
    start = time.time()
    n = len(arr)
    if n <= 1:
        return
    for i in range(1, n):
        key = arr[i]
        j = i - 1
        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
    end = time.time()
    return end - start

t = insertionSort(a)
print("Time taken for sorting an unsorted array:", t)

t = insertionSort(a)
print("Time taken for sorting an ascending sorted array:", t)
```

```
a.reverse()
t = insertionSort(a)
print("Time taken for sorting a descending sorted array:", t)
```

### Output:

```
PS C:\Users\Mugilkrishna D U\OneDrive\Desktop\My Files\.vscode> & "C:/Users/Mugilkrishna D U/AppData/Local/Programs/Python/Python311/python.exe" "c:/Users/Mugilkrishna D U/OneDrive/Desktop/My Files/SSN/SEM4/DESIGN AND ANALYSIS OF ALGORITHMS/LAB/1.3.py"
Enter the number of elements: 1000
Time taken for sorting an unsorted array: 0.03425407409667969
Time taken for sorting an ascending sorted array: 0.0
Time taken for sorting a descending sorted array: 0.09128713607788086
PS C:\Users\Mugilkrishna D U\OneDrive\Desktop\My Files\.vscode> & "C:/Users/Mugilkrishna D U/AppData/Local/Programs/Python/Python311/python.exe" "c:/Users/Mugilkrishna D U/OneDrive/Desktop/My Files/SSN/SEM4/DESIGN AND ANALYSIS OF ALGORITHMS/LAB/1.3.py"
Enter the number of elements: 10000
Time taken for sorting an unsorted array: 5.567095756530762
Time taken for sorting an ascending sorted array: 0.002999544143676758
Time taken for sorting a descending sorted array: 11.208353519439697
PS C:\Users\Mugilkrishna D U\OneDrive\Desktop\My Files\.vscode> & "C:/Users/Mugilkrishna D U/AppData/Local/Programs/Python/Python311/python.exe" "c:/Users/Mugilkrishna D U/OneDrive/Desktop/My Files/SSN/SEM4/DESIGN AND ANALYSIS OF ALGORITHMS/LAB/1.3.py"
Enter the number of elements: 100000
Time taken for sorting an unsorted array: 706.4955728054047
Time taken for sorting an ascending sorted array: 0.023004770278930664
Time taken for sorting a descending sorted array: 1301.9972875118256
```

**Time Complexity Analysis:** Time Complexity for Insertion Sort:  $O(n^2)$ ,  $\Omega(n)$ ,  $\Theta(n^2)$

### Time Complexity Analysis table:

- Ascending:  $O(1)$  – Best Case
- Descending:  $O(n^2)$  – Worst Case
- Not Sorted:  $O(n^2)$

No of elements	Not Sorted (Unsorted) (Time taken)	Ascending sorted (Time taken)	Descending Sorted (Time taken)
1000	0.03425	0.0	0.09128
10000	5.56709	0.00299	11.20835
100000	706.49557	0.02300	1301.99728



**Question 4:** Compare the performances of recursive and non-recursive algorithms for binary search using an array of size 100000.

**Algorithm:**

Algorithms:  
(i) Recursive:  
binarySearch(L, start, end, Key):  
    if end >= start:  
        mid = (start + end) / 2  
        if L[mid] == Key:  
            return mid  
        else if L[mid] < Key:  
            return binarySearch(L, mid + 1, end, Key)  
        else:  
            return binarySearch(L, start, mid - 1, Key)  
    else:  
        return -1  
(ii) Iterative / Non-recursive  
binarySearch(L, Key):  
    start = 0  
    end = len(L)  
    mid = 0  
    while start < end:  
        mid = (end + start) / 2  
        if L[mid] == Key:  
            return mid  
        else if L[mid] < Key:  
            start = mid + 1  
        else:  
            end = mid - 1  
    return -1

mid = (end + start) / 2  
if L[mid] == Key:  
    return mid  
else if L[mid] < Key:  
    start = mid + 1  
else:  
    end = mid - 1  
return -1

**Code:**

```
import random
import time

n = int(input("Enter the number of elements: "))
a = random.sample(range(1, 1000000), n)
```

```
a.sort()

# Recursive Binary Search
def binarySearch(L, low, high, key):
    # Measure start time
    start = time.time()
    if high >= low:
        mid = (low + high) // 2
        if L[mid] == key:
            # Measure end time if key is found
            end = time.time()
            return end - start
        elif L[mid] > key:
            return binarySearch(L, low, mid - 1, key)
        else:
            return binarySearch(L, mid + 1, high, key)
    else:
        # Measure end time if key is not found
        end = time.time()
        return end - start

element = random.randint(1, 1000000)

# Perform recursive binary search

time_recursive = binarySearch(a, 0, len(a) - 1, element)
print("Time taken for Recursive Binary Search:", time_recursive)

# Non-Recursive Binary Search
def binary_search(L, key):
    # Measure start time
    start = time.time()
    low = 0
    high = len(L) - 1
    while low <= high:
        mid = (high + low) // 2
        if L[mid] < key:
            low = mid + 1
        elif L[mid] > key:
            high = mid - 1
        else:
            # Measure end time if key is found
            end = time.time()
            return end - start

    # Measure end time if key is not found
    end = time.time()
```

```
    return end - start
# Perform non-recursive binary search
time_non_recursive = binary_search(a, element)
print("Time taken for Non-Recursive Binary Search:", time_non_recursive)
```

#### Output:

```
~/EdibleRespectfulDos$ python3 main.py
Enter the number of elements: 10
Time taken for Recursive Binary Search: 2.384185791015625e-07
Time taken for Non-Recursive Binary Search: 2.6226043701171875e-06
~/EdibleRespectfulDos$ python3 main.py
Enter the number of elements: 100
Time taken for Recursive Binary Search: 2.384185791015625e-07
Time taken for Non-Recursive Binary Search: 5.0067901611328125e-06
~/EdibleRespectfulDos$ python3 main.py
Enter the number of elements: 1000
Time taken for Recursive Binary Search: 2.384185791015625e-07
Time taken for Non-Recursive Binary Search: 5.7220458984375e-06
~/EdibleRespectfulDos$ python3 main.py
Enter the number of elements: 10000
Time taken for Recursive Binary Search: 2.384185791015625e-07
Time taken for Non-Recursive Binary Search: 7.3909759521484375e-06
~/EdibleRespectfulDos$ python3 main.py
Enter the number of elements: 100000
Time taken for Recursive Binary Search: 2.384185791015625e-07
Time taken for Non-Recursive Binary Search: 1.4781951904296875e-05
~/EdibleRespectfulDos$
```

#### Time Complexity Analysis Table:

For both recursive and non-recursive algorithms, the time complexity is  $O(\log n)$

No of elements	Recursive Binary Search (Time taken)	Non-Recursive Binary Search (Time taken)
10	2.38418e-07	2.62260e-06
100	2.38418e-07	5.00679e-06
1000	2.38418e-07	5.72204e-06
10000	2.38418e-07	7.39097e-06
100000	2.38418e-07	1.47819e-05

#### Learning Outcome:

Upon completing this exercise, I've gained insights into various sorting techniques and observed how their execution times vary with input size. Additionally, I've grasped the execution times of both recursive and non-recursive binary search algorithms when applied to a randomly generated unique array.