INSTITUTE OF COMPUTER TECHNOLOGY

B-TECH COMPUTER SCIENCE ENGINEERING 2025-26

SUBJECT: ALGORITHM ANALYSIS & DESIGN

NAME: Rahul Prajapati

ENRLL. NO: 23162171020

BRANCH: CYBER SECURITY

BATCH: 52

PRACTICAL_03

AIM: NextMid Technology is an American food company that manufactures, markets, and distributes spices, seasoning mixes, condiments, and other flavoring products for the industrial, restaurant, institutional, and home markets, they are having some number quantity of different categories item food, kindly help them to sort data using any three sorting methods and determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted and plot a graph of the comparison between them. Design the algorithm for the same and implement using the programming language of your choice. Make comparative analysis for various use cases & input size.

Questions:

What is the best, average and worst case analysis of algorithms?

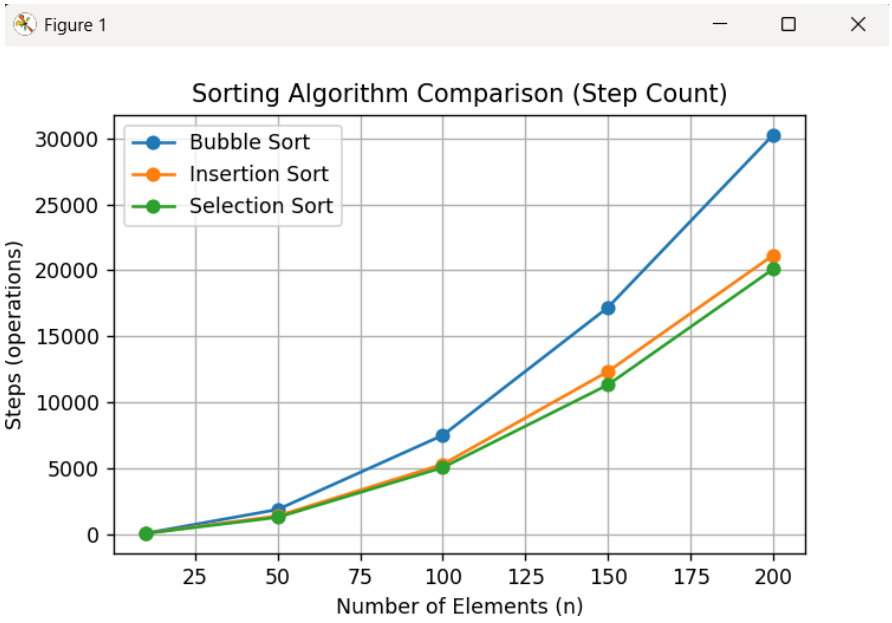
Which are different asymptotic notations? What is their use?

What is the time complexity of above 3 sorting algorithms in all cases?

Code:

```python
import random
import matplotlib.pyplot as plt

# Bubble Sort
def bubble_sort(arr):
    steps = 0
    n = len(arr)
    for i in range(n):
        for j in range(0, n - i - 1):
            steps += 1
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
                steps += 1
    return steps

#Insertion Sort
def insertion_sort(arr):
    steps = 0
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        steps += 1
        while j >= 0 and arr[j] > key:
            steps += 1
            arr[j + 1] = arr[j]
            steps += 1
            j -= 1
        arr[j + 1] = key
        steps += 1
    return steps

#Selection sort
def selection_sort(arr):
    steps = 0
    n = len(arr)
    for i in range(n):
        min_index = i
        for j in range(i + 1, n):
            steps += 1
            if arr[j] < arr[min_index]:
                min_index = j
        arr[i], arr[min_index] = arr[min_index], arr[i]
        steps += 1
    return steps

sizes = [10, 50, 100, 150, 200]
b_steps = []
i_steps = []
s_steps = []

for size in sizes:
    data = []
    for k in range(size):
        data.append(random.randint(1, 500))
    print(f"data{size}:{data}")
    b_steps.append(bubble_sort(data.copy()))
    i_steps.append(insertion_sort(data.copy()))
    s_steps.append(selection_sort(data.copy()))

plt.plot(sizes, b_steps, label="Bubble Sort", marker='o')
plt.plot(sizes, i_steps, label="Insertion Sort", marker='o')
plt.plot(sizes, s_steps, label="Selection Sort", marker='o')

plt.xlabel("Number of Elements (n)")
plt.ylabel("Steps (operations)")
plt.title("Sorting Algorithm Comparison (Step Count)")
plt.legend()
plt.grid(True)
plt.show()
```

Output:



1. What is the best, average and worst case analysis of algorithms?

1. Bubble Sort

- **Best Case: O(n)**
  - Happens when the list is already sorted.
  - Only one pass is needed to check that no swaps are required.
- **Average Case: O(n²)**
  - Happens when the list is randomly arranged.
  - On average, about half of the elements are out of place.
- **Worst Case: O(n²)**
  - Happens when the list is in reverse order.
  - Needs maximum number of swaps.

2. Insertion Sort

- **Best Case: O(n)**
  - Happens when the list is already sorted.
  - Only one comparison per element is made.
- **Average Case: O(n²)**
  - Happens when the list is randomly ordered.
  - Each new element may need to be compared with about half of the sorted part.
- **Worst Case: O(n²)**
  - Happens when the list is in reverse order.
  - Every new element must be compared and shifted all the way to the beginning.

3. Selection Sort

- **Best Case: O(n²)**
  - Even if the list is sorted, selection sort still scans the entire remaining list to find the minimum.
- **Average Case: O(n²)**
  - Always scans the remaining elements for each position, no matter how sorted it is.
- **Worst Case: O(n²)**
  - Same reason as above — it always goes through all elements to find the minimum.

2. Which are different asymptotic notations? What is their use?

1. Big-O (O)

- **Short Description:** Shows the **slowest speed** an algorithm can have — the most steps it might take.
- **Example:** Bubble Sort can take **$O(n^2)$** steps if the list is in the worst order.
- **Why We Use It:** So we know it will never be slower than this limit.

2. Big-Omega (Ω)

- **Short Description:** Shows the **fastest speed** an algorithm can have — the least steps it might take.
- **Example:** Insertion Sort takes **$\Omega(n)$** steps if the list is already sorted.
- **Why We Use It:** So we know how good it can be in the best case.

3. Big-Theta (Θ)

- **Short Description:** Shows the **normal speed** — when best and worst cases are about the same.
- **Example:** Selection Sort always takes **$\Theta(n^2)$** steps no matter what.
- **Why We Use It:** So we understand its usual performance.

3. What is the time complexity of above 3 sorting algorithms in all cases?

- Bubble Sort –
  - Best: O(n) : if already sorted
  - Average/Worst: $O(n^2)$
- Insertion Sort –
  - Best: O(n) if already sorted
  - Average/Worst: $O(n^2)$
- Selection Sort –
  - Best/Average/Worst: $O(n^2)$ - always scans full list either sorted or not.