**Add a Cover Page before this page and add the lab report number also.**

**Problem Statement:** Stepwise Execution Analysis of Sorting Algorithms I

**Bubble Sort**

**Theory:** Just write down the algorithm step-by-step here. An addition of flowchart of the algorithm is a must.Everyone is suggested to follow this format: the whole report should be written in **Times New Roman** font; the font size is 12. Only the Title should be of 20 font size. The alignment of you report should be justified. To justify your paragraph. Select your paragraph and press ctrl + j.

**Code:** In **code** section, please use this website [https://syntax-highlighter.k26.ch/#](https://syntax-highlighter.k26.ch/) to highlight your code. All you need to do is copy your code from your editor and paste on the website mentioned above… Then click copy highlighted button or just copy the highlighted code from there and paste here…

After pasting, your code should look like this:

1. #include <iostream>
2. **using** **namespace** std;
4. **int** findSquareRoot(**int** n) {
5. **int** start = 0, end = n, ans = 0;
7. **while** (start <= end) {
8. **int** mid = start + (end - start) / 2;
10. **if** (mid \* mid == n) {
11. **return** mid; // Exact square root found
12. } **else** **if** (mid \* mid < n) {
13. ans = mid; // Update answer
14. start = mid + 1;
15. } **else** {
16. end = mid - 1;
17. }
18. }
20. **return** ans; // Return the largest integer x such that x^2 <= n
21. }
23. **int** main() {
24. **int** n;
25. cout << "Enter a non-negative integer: ";
26. cin >> n;
28. **int** result = findSquareRoot(n);
29. cout << "The largest integer x such that x^2 <= " << n << " is: " << result << endl;
31. **return** 0;
32. }

**Screenshots:** After adding the code, you have to add the screenshot of the code as well as the code output.

**Analysis:** Here, you have to trace the algorithm line by line to see how it processes data and arrives at the result. An example for Bubble Sort:

Input Array: A = [5, 3, 4, 1, 2]

Stepwise Execution (Pass by Pass)

Pass 1 (i = 1):

Compare 5 and 3 → swap → [3, 5, 4, 1, 2]

Compare 5 and 4 → swap → [3, 4, 5, 1, 2]

Compare 5 and 1 → swap → [3, 4, 1, 5, 2]

Compare 5 and 2 → swap → [3, 4, 1, 2, 5]

Pass 2 (i = 2):

Compare 3 and 4 → no swap → [3, 4, 1, 2, 5]

Compare 4 and 1 → swap → [3, 1, 4, 2, 5]

Compare 4 and 2 → swap → [3, 1, 2, 4, 5]

Pass 3 (i = 3):

Compare 3 and 1 → swap → [1, 3, 2, 4, 5]

Compare 3 and 2 → swap → [1, 2, 3, 4, 5]

Pass 4 (i = 4):

Compare 1 and 2 → no swap → [1, 2, 3, 4, 5]

Final Sorted Array: [1, 2, 3, 4, 5]

**Insertion Sort**

**Theory:** Same instruction as before.

**Code:** Same instruction as before.

**Screenshots:** Same instruction as before.

**Analysis:** Same instruction as before.

**Selection Sort**

**Theory:** Same instruction as before.

**Code:** Same instruction as before.

**Screenshots:** Same instruction as before.

**Analysis:** Same instruction as before.

**Observation Table**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Test Case No. | Input Size (n) | Number of Comparisons | | | Number of Swap/Shift | | | Execution Time | | | |
| Bubble Sort | Insertion Sort | Selection Sort | Bubble Sort | Insertion Sort | Selection Sort | Bubble Sort | Insertion Sort | Selection Sort |
| 01 | 10 (Best) | 9 | 0 | 45 | 0 | 0 | 0 | 4.012 s | 9.287 s | 9.454 s |
| 02 | 10 (Average) | 44 | 19 | 45 | 19 | 19 | 15 | 6.629 s | 7.971 s | 7.907 s |
| 03 | 10 (Worst) | 45 | 45 | 45 | 45 | 45 | 45 | 2.387 s | 9.258 s | 7.548 s |
| 04 | 100 (Best) | 99 | 0 | 4950 | 0 | 0 | 0 | 2.936 s | 9.159 s | 7.325 s |
| 05 | 100 (Average) | 4814 | 2404 | 4950 | 2404 | 2404 | 1490 | 7.265 s | 9.638 s | 7.465 s |
| 06 | 100 (Worst) | 4950 | 4950 | 4950 | 4950 | 4950 | 4950 | 4.136 s | 9.250 s | 8.598 s |
| 07 | 1000 (Best) | 999 | 0 | 499500 | 0 | 0 | 0 | 5.044 s | 8.964 s | 6.935 s |
| 08 | 1000 (Average) | 498465 | 241331 | 499500 | 241331 | 241331 | 171500 | 4.063 s | 6.799 s | 8.547 s |
| 09 | 1000 (Worst) | 499500 | 499500 | 499500 | 499500 | 499500 | 499500 | 3.203 s | 6.358 s | 8.487 s |

Conclusion:

Bubble Sort, Insertion Sort and Selection Sort are best suited for small datasets, but their efficiency varies with data conditions. **Bubble Sort** works well only for very small or nearly sorted data but becomes highly inefficient as input size grows. **Insertion Sort** is the most efficient among the three for **small or nearly sorted datasets**, offering faster execution with fewer comparisons and shifts. **Selection Sort**, though predictable with a fixed number of comparisons, is slower on nearly sorted data and less efficient for large inputs. For practical purposes, **Insertion Sort** is generally the preferred choice for small datasets, while Bubble and Selection Sort are mostly useful for educational demonstrations or very limited data sizes.

Remember that the lab reports you’re writing now will be really helpful in your exams and viva. So, I suggest you to add information that you genuinely find useful and avoid copying other’s lab report.