Arrays Question with Solutions

1. **What happens if the binary search is applied to an element that occurs more than once in the array?**

If the binary search is applied to an element that occurs more than once in an array, it could return the index of any one of them. It depends upon how close their indexes are to multiples of midpoints of subintervals. For example, if the binary search is applied in an array of 10,000 elements, searching for a value that is repeated at locations 0–99, the search would return the index 77 on the 7th iteration.

2. **If the sequential search took 50 ms to run on an array of 10,000 elements, how long would you expect it to take to run on an array of 20,000 elements on the same computer?**

The sequential search runs in linear time, which means that the time is proportional to the number of elements. So an array with twice as many elements would take twice as long to process: 20 ms

3. **What is the simplest way to print an array of objects?**

The simplest way to print an array of objects is to pass it to the Arrays.toList() method which produces a List object that can be printed directly with the System.out.println() method.

4. **What happens if the sequential search is applied to an element that occurs more than once in the array?**

If the sequential search is applied to an element that occurs more than once in an array, it will return the index of the one that is closest to the beginning of the array.

5. **Why Time Complexity of an Algorithm is important?**

- To estimate how long a program will run
- To estimate the largest input that can reasonably be given to the program
- To compare the efficiency of different algorithms
- To help focus on the parts of code which are executed the largest number of times

6. **Compute the running time of following algorithm**

```
1. n = read input from user
2. sum = 0
3. i = 0
4. while i < n
5.       number = read input from user
6.       sum = sum + number
7.       i = i + 1
8. End while
9. mean = sum / n
```

Solution

Statement executed	Number of times
1	1
2	1
3	1
4	n+1
5	n
6	n
7	n
8	0
9	1

The computing time for this algorithm in terms of input size n is: $T(n) = 4n + 5$.

7. **Calculate the running time of following algorithm**

```
for i = 1 to n { // assume that n is input size
        ...
        for j = 1 to 2*i {
            ...
            k = j;
                while (k >= 0) {
                    ...
                    k = k - 1;
                                }
                            }
        }
```

Solution

Let $I()$, $M()$, $T()$ be the running times for (one full execution of) the inner loop, middle loop, and the entire program. We work from inside out.

The number of passes through the loop depends on j. It is executed for $k = j, j-1, j-2, \ldots\ldots, 0$

$$I(j) = \sum_{k=0}^{j} 1 = j + 1$$

Now consider the Middle loop

- Running time is determined by i

$$M(i) = \sum_{j=1}^{2i} I(j) = \sum_{j=1}^{2i} (j + 1).$$

- And we know that $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$. $\quad M(i) = \sum_{j=1}^{2i} j + \sum_{j=1}^{2i} 1.$

We get $M(i) = \frac{2i(2i+1)}{2} + 2i = \frac{4i^2 + 2i + 4i}{2} = 2i^2 + 3i.$

Now for the outermost loop we have

$$T(n) = \sum_{i=1}^{n} (2i^2 + 3i). \qquad T(n) = 2\sum_{i=1}^{n} i^2 + 3\sum_{i=1}^{n} i.$$

The **Sum of Squares Formula** for N values is given as,

$$1^2 + 2^2 + 3^2 + ... + n^2 = \frac{n(n+1)(2n+1)}{6}$$

$$\sum_{i=1}^{n} i^2 = 1 + 4 + 9 + \cdots + n^2 = \frac{2n^3 + 3n^2 + n}{6}.$$

For n >= 0 $T(n) = 2\frac{2n^3 + 3n^2 + n}{6} + 3\frac{n(n+1)}{2},$

So $T(n) = \frac{4n^3 + 15n^2 + 11n}{6}.$

or simply $T(n^3)$

8. What are the challenges of the experimental analysis? What are the points to be kept into consideration to develop an approach to analyze the efficiency of an algorithm?

Solution

While experimental studies of running times are valuable, especially when fine-tuning production-quality code, there are three major limitations to their use for algorithm analysis:

- Experimental running times of two algorithms are difficult to directly compare unless the experiments are performed in the same hardware and software environments.
- Experiments can be done only on a limited set of test inputs; hence, they leave out the running times of inputs not included in the experiment (and these inputs may be important).
- An algorithm must be fully implemented in order to execute it to study its running time experimentally.

This last requirement is the most serious drawback to the use of experimental studies. At early stages of design, when considering a choice of data structures or algorithms, it would be foolish to spend a significant amount of time implementing an approach that could easily be deemed inferior by a higher-level analysis.

Our goal is to develop an approach to analyzing the efficiency of algorithms that:

1. Allows us to evaluate the relative efficiency of any two algorithms in a way that is independent of the hardware and software environment.
2. Is performed by studying a high-level description of the algorithm without need for implementation.
3. Takes into account all possible inputs.
