Search Test Lab Report

Names: Yitian Xu

**1. Linear Search**

We know from class that the theoretical time complexity of linear search over *unordered lists* is:

|  |  |  |
| --- | --- | --- |
| **Best Case** | **Worst Case** | **Average Case** |
| *1* | *N* | *N/2* |

**Q1:** Increasing the number of trials and the value of N

1. Run experiments with an increasing value of N (from 1000 to 10,000). Does increasing N affect how many trials you have to run to get accurate results? Explain.

Answer:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | Trial | best (1) | wrost (N) | avg (N/2) |
| 1000 | 10 | 98 | 897 | 572.9 |
| 1000 | 100 | 4 | 968 | 443.04 |
| 1000 | 500 | 1 | 1000 | 475.928 |
| 1000 | 1000 | 1 | 1000 | 504.469 |
| 5000 | 10 | 308 | 4933 | 2713 |
| 5000 | 100 | 23 | 4957 | 2566.52 |
| 5000 | 500 | 4 | 4998 | 2533.274 |
| 5000 | 1000 | 2 | 4995 | 2543.757 |
| 10000 | 10 | 1017 | 9972 | 6584.2 |
| 10000 | 100 | 168 | 9757 | 5009.2 |
| 10000 | 500 | 9 | 9995 | 4592.768 |
| 10000 | 1000 | 6 | 10000 | 4931.94 |

As shown above, we can easily see that in range:(1000,10000), it is always not accurate until trial reach 500. So I don’t think increasing N works.

Besides, the time of sorting an unordered list completely depends on the very list of numbers that passed to the function. If it is more-ordered like, the time increases, vice versa. So to get close to the theoretical value.

1. Write down the number of trials that seem to have worked well for N=10,000.

|  |
| --- |
| **Number of Trials** |
| 1000 |

**Q2:** Linear Search Time Complexity Plot (Unordered List)

|  |
| --- |
|  |

**Q3:** Does the order of the data in the list affect the number of comparisons? In the table below, guess the time complexity of Linear Search on an *Ordered List.*

|  |  |  |
| --- | --- | --- |
| **Best Case** | **Worst Case** | **Average Case** |
| 1 | N | N/2 |

Linear Search Time Complexity Plot (Ordered List)

|  |
| --- |
|  |

**Conclusion:**

Whether the list is sorted or not has no impact on time complexity of selection sort.

**2. Binary Search**

We know from class that the theoretical time complexity of binary search over *ordered lists* are:

|  |  |  |
| --- | --- | --- |
| **Best Case** | **Worst Case** | **Average Case** |
| *1* | *log\_2(N)* | *???* |

**Q4:** Binary Search Time Complexity Plot

|  |
| --- |
|  |

**Conclusion:** What do your results tell you about the average-case complexity of Binary Search?

Answer: It is somewhere near the wrost case.

**3. Median**

Q5: We hypothesize that the time complexity of find\_median is:

|  |  |  |
| --- | --- | --- |
| **Best Case** | **Worst Case** | **Average Case** |
| N | N^2 | (N^2 + N) / 2 |

**Justification:**

1. Best case scenario:

*Happens when the median is the first element of the list.*

1. Wrost case scenario:

*Happens when the median is the last element of the list.*

1. Average case scenario:

Assume the distribution of the median uniform-distribution. Then we can calculate the average case using (N + N^2) / 2.

Find\_median Time Complexity Plot

|  |
| --- |
|  |

**Conclusion:** Did your results support your hypothesis? If not, why not, and how does it change your original hypothesis?

Yes! Even in the best scenario, you still have to iterate the whole list.