# Project Report

## Comparison between Multiprocessing and Multithreading via Sorting and Searching


## By:

Masir Ali Khan 22k-8702
Ibadur Rahman 22k-4064
Arish 22k-4118
Ahsan 22k-4021

# Introduction

The purpose of this project is to differentiate between multithreading and multiprocessing, by measuring the memory and time taken to complete various sorting and searching algorithms using C language on Ubuntu.

# Methodology

An array will be used for the same type of sorting and searching algorithms, where the memory and time taken will be measured for each of the codes.

# Algorithms

The following algorithms will be utilized:

1. Binary Search
2. Interpolation Search
3. Heap Sort
4. Quick Sort
5. Merge Sort

# Results

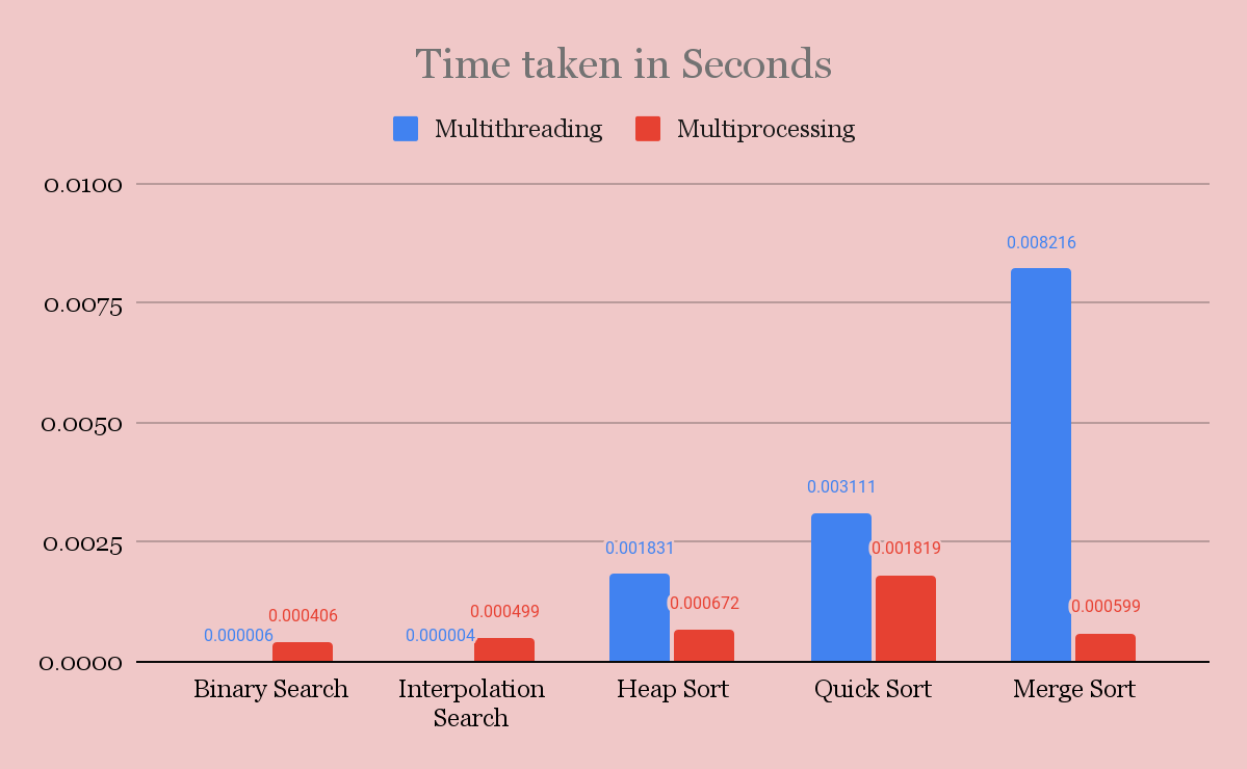
## Time (Seconds):

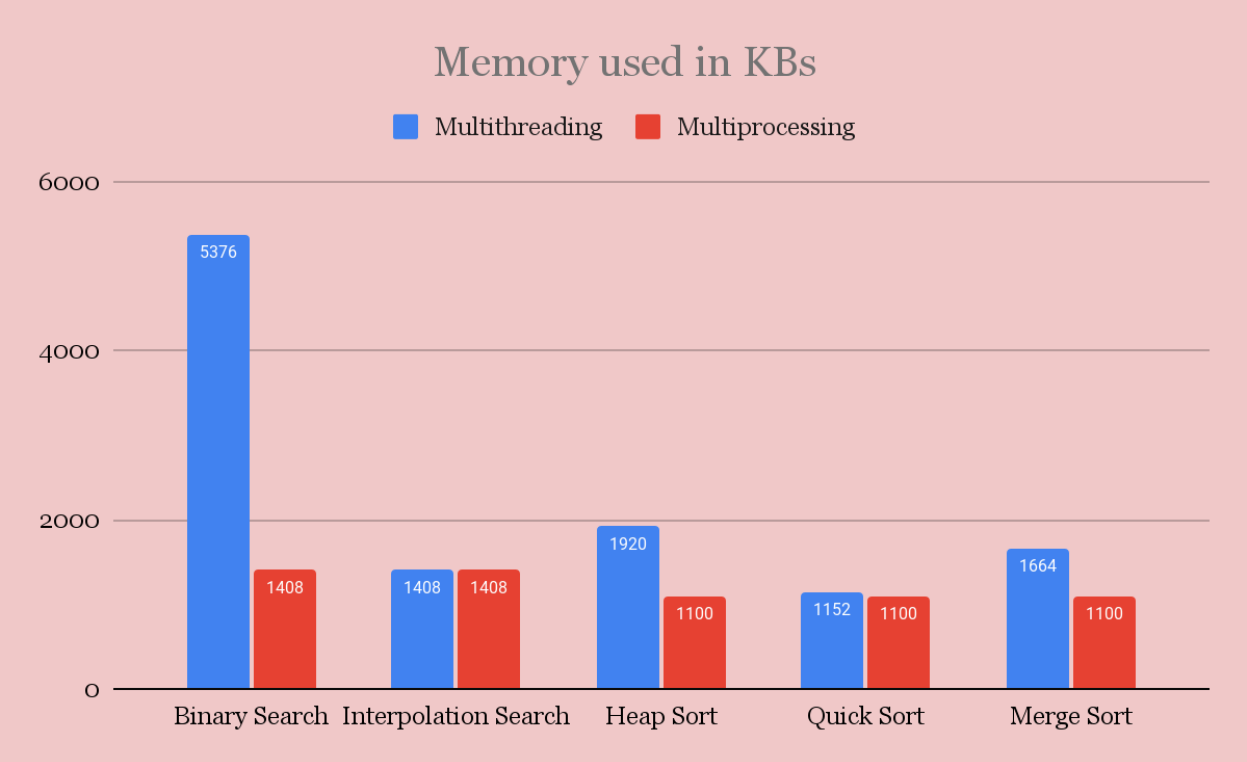
| | Multithreading | Multiprocessing |
|---|---|---|
| Binary Search | 0.000006 | 0.000406 |
| Interpolation Search | 0.000004 | 0.000499 |
| Heap Sort | 0.001831 | 0.000672 |
| Quick Sort | 0.003111 | 0.001819 |
| Merge Sort | 0.008216 | 0.000599 |

## Memory (KBs):

| | Multithreading | Multiprocessing |
|---|---|---|
| Binary Search | 5376 | 1408 |
| Interpolation Search | 1408 | 1408 |
| Heap Sort | 1920 | 1100 |
| Quick Sort | 1152 | 1100 |
| Merge Sort | 1664 | 1100 |

# Comparison


Time taken in Seconds
Multithreading
Multiprocessing
0.0100
0.0075
0.0050
0.0025
0.0000
0.000006
0.000406
0.000004
0.000499
0.001831
0.000672
0.003111
0.001819
0.008216
0.000599
Binary Search
Interpolation Search
Heap Sort
Quick Sort
Merge Sort


Memory used in KBs
Multithreading
Multiprocessing
6000
4000
2000
0
5376
1408
1408
1408
1920
1100
1152
1100
1664
1100
Binary Search
Interpolation Search
Heap Sort
Quick Sort
Merge Sort

# Conclusion

**The summary of our tests proves the following:**

1. Multithreading is faster when using searching algorithms, while consuming the same memory. (Binary search is an anomaly because we were printing the result of each iteration individually, and then adding the sum, producing more overhead).
2. Multiprocessing provides results much quicker when it comes to sorting algorithms.
3. Multiprocessing uses less memory than multithreading for both sorting and searching algorithms.

**Reasons for Results:**

1. Multithreading excels in searching algorithms due to its ability to efficiently handle concurrent tasks, enabling faster execution without significantly increasing memory consumption.
2. Sorting algorithms benefit from multiprocessing's parallel processing capabilities, leading to faster results by leveraging multiple CPU cores simultaneously.
3. In comparison to multithreading, multiprocessing typically utilizes memory more efficiently for both sorting and searching algorithms, likely due to separate memory spaces for each process, reducing memory overhead.