Experiment 1: Quicksort Pivot Selection

**Algorithmic choices:**

1. C++ sort
2. Quicksort

**Pivot selection:**

* 1. A[hi] as the pivot
  2. Random element in the array as the pivot

**Type of input:**

1. Reverse order
2. Random input

**Table and plot:**

Reverse order

|  |  |  |  |
| --- | --- | --- | --- |
|  | C++ sort | Quicksort ( A[hi] ) | Quicksort ( A[Random i] ) |
| 10000 | 0.5401 | 143.14 | 0.7999 |
| 20000 | 1.0992 | 563.666 | 1.6524 |
| 30000 | 1.584 | 1276.46 | 2.3829 |
| 40000 | 2.299 | 2249.77 | 3.3065 |
| 50000 | 2.7613 | 3500.71 | 4.1759 |

The plot blow shows the reverse order input with C++ sort and Quicksort A[hi] as the pivot and Random element in the array as the pivot

A screenshot of a cell phone

Description automatically generated

Random order

|  |  |  |  |
| --- | --- | --- | --- |
|  | C++ sort | Quicksort ( A[hi] ) | Quicksort ( A[Random i] ) |
| 10000 | 1.4566 | 1.0627 | 1.186 |
| 20000 | 3.1771 | 2.2487 | 2.3773 |
| 30000 | 4.916 | 3.5607 | 3.8461 |
| 40000 | 6.7542 | 4.9118 | 5.0399 |
| 50000 | 8.4948 | 6.034 | 6.3359 |

The plot blow shows the random order input with C++ sort and Quicksort A[hi] as the pivot and Random element in the array as the pivot

A picture containing map, text, air

Description automatically generated  
**observation:**

Reverse order input with Quicksort A[hi] as pivot is extremely time consuming compare to the other five plots. It is because this is the worst scenario for quicksort with O(n^2) time complexity. C++ sort is more stable compare to quicksort with different input, and it is faster than the best scenario(approximately O(nlog(n))) in the four quicksort method with random input and random element as the pivot.

**purpose of source code file:**

experiment1.h: contains quicksort method with hi and random as pivot

test1.cpp: test the timing for each sorting method

Experiment 2: IntroSort

**Algorithmic choices:**

1. C++ sort
2. standard Quicksort with middle as pivot
3. modified Quicksort with middle as pivot
4. Insertion Sort

**Type of input:**

1. Reverse order
2. Random input

**Table and plot:**

Reverse order

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | C++ sort | Standard Quicksort | modified Quicksort | Insertion Sort |
| 8 | 0.0012 | 0.0009 | 0.001 | 0.0008 |
| 9 | 0.0012 | 0.001 | 0.0008 | 0.0008 |
| 10 | 0.0012 | 0.001 | 0.0008 | 0.0009 |
| 11 | 0.0013 | 0.0009 | 0.0009 | 0.001 |

The plot blow shows the reverse order input with C++ sort, standard Quicksort, modified Quicksort and Insertion Sort.

A screenshot of a map

Description automatically generated

Random order

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | C++ sort | Standard Quicksort | modified Quicksort | Insertion Sort |
| 55 | 0.0053 | 0.0037 | 0.0028 | 0.0031 |
| 60 | 0.0068 | 0.0057 | 0.0042 | 0.0047 |
| 65 | 0.0055 | 0.0041 | 0.0034 | 0.0038 |
| 70 | 0.0068 | 0.0048 | 0.0036 | 0.005 |

The plot blow shows the Random order input with C++ sort, standard Quicksort, modified Quicksort and Insertion Sort.

A close up of a map

Description automatically generated

**observation:**

S is the intersection of Insertion sort curve and standard quicksort curve. Insertion sort is faster than standard Quicksort when the size is less than S, but slower than standard Quicksort when the size is greater than S. For reverse order input, S is 10, for random order input, S is 65.

**purpose of source code file:**

experiment2.h: contains standard and modified quicksort method with middle as pivot

test2.cpp: test the timing for each sorting method