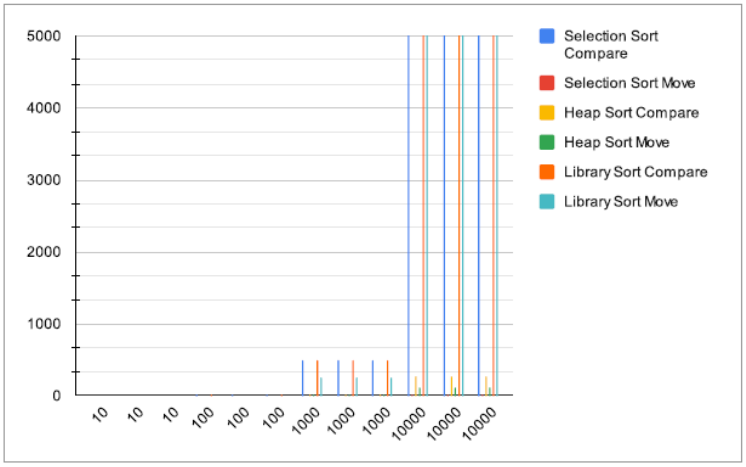
Sorting Analysis

Mack Bautista
201729981
Bita Sadeghi
Assignment 5
Mar. 8, 2024

Summary

The following table shows the chosen three algorithms: selection sorting, heap sorting, and library sorting. Each array size has a total of three tests in which the selection sorting requires the most linear results. In each array size, the selection sort always has the same comparison count, and move count. Meanwhile heap sorting took the least amount of counts as its array size increases.

Array Size	ɛction Sort Comp	lection Sort Mov	eap Sort Compa	Heap Sort Move	rary Sort Comp	ɛibrary Sort Move
10	45	9	31	23	57	25
10	45	9	40	27	55	23
10	45	9	43	28	33	12
100	4950	99	1148	579	4545	2224
100	4950	99	1175	592	4188	2045
100	4950	99	1125	567	5088	2496
1000	499500	999	19683	9057	492055	245530
1000	499500	999	19592	9021	496010	247508
1000	499500	999	19593	9035	493545	246274
10000	49995000	9999	277672	123356	49639705	24814853
10000	49995000	9999	276878	123123	49907721	24948863
10000	49995000	9999	277521	123385	49857303	24923654



The following graph only displays the most significant results when the array size reaches past 1000. Out of all the sorting algorithms, heap sorting took the least amount of time. Library sorting is all over the place as it took the most time sorting. In conclusion, heap sorting is the most efficient algorithm out of all the three. As

the data sizes increase, heap sorting will reduce the amount of time the data gets sorted which is overall better at managing time.

Java Code

```
package SortingCompare;
public class SortingAnalysis {

      public static final int[] INPUTSIZES = {10000};

   static SelectionSort selection = new SelectionSort();
   static HeapSort heap = new HeapSort();
   static LibrarySort library = new LibrarySort();

      public static int[] generateRandomArray(int size) {
       int[] arr = new int[size];
       for (int i = 0; i < size; i++) {
           arr[i] = (int) (Math.random() * 100);
       }
       return arr;
   }

   public static void printArray(int[] arr) {
       for (int num : arr) {
           System.out.print(num + " ");
       }
       System.out.println();
   }

   public static void runSelection(int[] arr) {
           System.out.println("Original array: ");
           //printArray(arr);

           selection.selectionSort(arr);

           System.out.println("Selection Sort array:");
           //printArray(arr);
                                System.out.println("Comparison   Count:   "   +
selection.getCompareCount());
                             System.out.println("Data   Movement   Count:   "   +
selection.getMoveCount());
           System.out.println();

           selection.setCompareCount(0);
           selection.setMoveCount(0);
   }
```

```java
public static void runHeap(int[] arr) {
    System.out.println("Original array: ");
    //printArray(arr);

    heap.heapSort(arr);

    System.out.println("Heap Sort array:");
    //printArray(arr);
    System.out.println("Comparison Count: " + heap.getCompareCount());
    System.out.println("Data Movement Count: " + heap.getMoveCount());
    System.out.println();

    heap.setCompareCount(0);
    heap.setMoveCount(0);
}

public static void runLibSort(int[] arr) {
    System.out.println("Original array: ");
    //printArray(arr);

    library.librarySort(arr);

    System.out.println("Library Sort array:");
    //printArray(arr);
    System.out.println("Comparison Count: " + library.getCompareCount());
    System.out.println("Data Movement Count: " + library.getMoveCount());
    System.out.println();

    library.setCompareCount(0);
    library.setMoveCount(0);
}


   public static void main(String[] args) {
    for (int size : INPUTSIZES) {
        int[] arr = generateRandomArray(size);

        int[] arrSelection = arr.clone();
        int[] arrHeap = arr.clone();
        int[] arrLibSort = arr.clone();

        runSelection(arrSelection);
        runHeap(arrHeap);
        runLibSort(arrLibSort);
    }
   }
```

```
}
```

```
package SortingCompare;
public class SelectionSort extends SortingAnalysis {
      private int compareCount = 0;
   private int moveCount = 0;
   public int getCompareCount() {
       return compareCount;
   }
   public void setCompareCount(int count) {
       compareCount = count;
   }
   public int getMoveCount() {
       return moveCount;
   }
   public void setMoveCount(int count) {
       moveCount = count;
   }

   public void selectionSort(int[] arr) {
       int n = arr.length;
       for (int i = 0; i < n - 1; i++) {
           int minIndex = i;
           for (int j = i + 1; j < n; j++) {
               compareCount++;
               if (arr[j] < arr[minIndex]) {
                   minIndex = j;
               }
           }
           int temp = arr[i];
           arr[i] = arr[minIndex];
           arr[minIndex] = temp;
           moveCount++;
       }
   }
}
package SortingCompare;
public class HeapSort extends SortingAnalysis {
   private int compareCount = 0;
   private int moveCount = 0;

   public int getCompareCount() {
       return compareCount;
   }
   public void setCompareCount(int count) {
       compareCount = count;
   }
```

```java
    public int getMoveCount() {
        return moveCount;
    }
    public void setMoveCount(int count) {
        moveCount = count;
    }
    public void heapSort(int[] arr) {
        int n = arr.length;
        for (int i = n / 2 - 1; i >= 0; i--) {
            heapify(arr, n, i);
        }
        for (int i = n - 1; i > 0; i--) {
            int temp = arr[0]; // swap roots with the largest element
            arr[0] = arr[i];
            arr[i] = temp;
            moveCount++;


            heapify(arr, i, 0); // heapify the reduced heap
        }
    }
    private void heapify(int[] arr, int n, int rootIndex) {
        int largest = rootIndex;
        int leftChild = 2 * rootIndex + 1;
        int rightChild = 2 * rootIndex + 2;
        if (leftChild < n && arr[leftChild] > arr[largest]) {
            largest = leftChild;
            compareCount++;
        }
        if (rightChild < n && arr[rightChild] > arr[largest]) {
            largest = rightChild;
            compareCount++;
        }
        if (largest != rootIndex) {
            int temp = arr[rootIndex];
            arr[rootIndex] = arr[largest];
            arr[largest] = temp;
            moveCount++;
            compareCount++;
            heapify(arr, n, largest); // heapify the affected subtree
        }
    }
}
package SortingCompare;
public class LibrarySort extends SortingAnalysis {
    private int compareCount = 0;
    private int moveCount = 0;

    public int getCompareCount() {
```

```
        return compareCount;
    }
    public void setCompareCount(int count) {
        compareCount = count;
    }
    public int getMoveCount() {
        return moveCount;
    }
    public void setMoveCount(int count) {
        moveCount = count;
    }

    public void librarySort(int[] arr) {
        int n = arr.length;
        int index = 0;
        while (index < n) {
            if (index == 0 || arr[index] >= arr[index - 1]) {
                index++;
            } else {
                // swap arr[index] with arr[index - 1]
                int temp = arr[index];
                arr[index] = arr[index - 1];
                arr[index - 1] = temp;
                moveCount++;
                if (index > 1) {
                    index--; // compare with the previous element
                }
            }
            compareCount++;
        }
    }


}
```