Project Heapsort

ANOUD AL-SUBAIE

SHAHAD AL-SAQABI

LEAN ALTWAYAN

Table of Contents

Description 2
Plot : 3
Pseudo-code: 4
Time complexity analysis for the algorithms 6
screen shots: 6

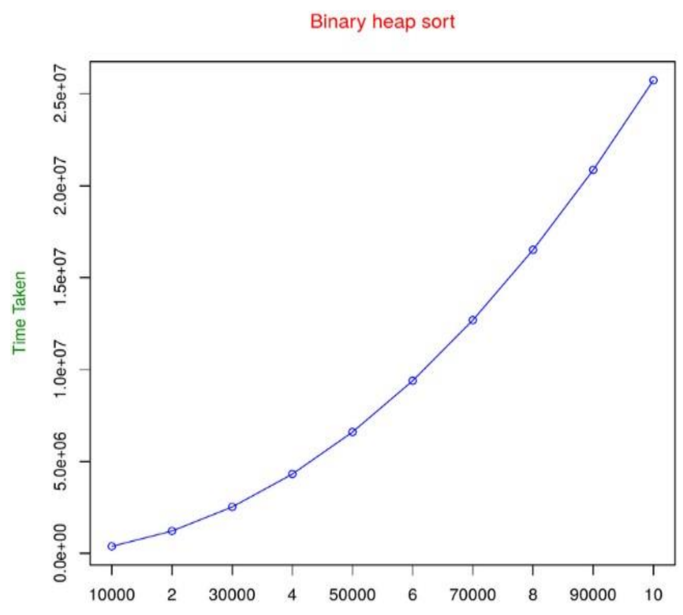
Description :

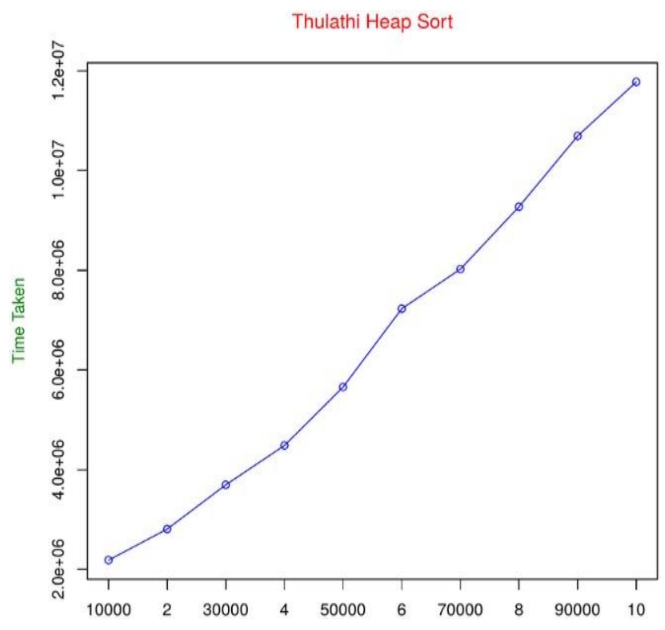
For this project we were asked to design a Thulathi heapsort algorithm and compare it with a binary heapsort regarding runtime and number of comparisons for each algorithm.

We were able to complete this task by first implementing both algorithms in java code, then giving both algorithms identical arrays and comparing their efficiency.

Finally, we plotted the binary and thulathi heapsorts to compare execution time vs array size using RStudio and R language

Plot :





Code for generating the plot :

```
# Define 2 vectors
time_taken <- c(2185848, 2809074, 3696687, 4485109,
5657532,7226719,8020976,9272168,10690380,11774187)
# Calculate range from 0 to max value
g_range <- range(0, 100000)
# Graph autos using y axis that ranges from 0 to max
# value in cars or trucks vector. Turn off axes and
# annotations (axis labels) so we can specify them ourself
plot(time_taken, type="o", col="blue", ylim=c(2185848,11774187), ann=FALSE)
# Make x axis using 10000-1000000 labels
axis(1, at=1:10,
lab=c("10000","20000","30000","40000","50000","60000","70000","80000","90000",
"100000"))
# Create box around plot
box()
# Create a title with a red, bold/italic font
title(main="Thulathi Heap Sort", col.main="red", font.main=1000000)

# Label the x and y axes with dark green text
title(xlab="Array size", col.lab=rgb(0,0.5,0))
title(ylab="Time Taken", col.lab=rgb(0,0.5,0))
```

Pseudo-code:

Binary Heap Sort :

```
BinaryHeapSort( arr [0.........r])
n ← r
for i = n/2    to   0
sort(arr , n , i)
for i = n-1   to   0
    temp ← arr[0]
    arr[0]← arr[i]
    arr[i]← temp
    sort( arr , n , 0)
```

Sort:

```
sort( arr[0....r], n, i)
largest ← i
L ← ( 2 * i) + 1            //left child
r ← ( 2 * i) + 2            //right child
if (L < n and arr[L] > arr[largest] )
      largest = L
if ( r < n and arr[r] > arr[Largest] )
      largest = r
if ( largest != i)
      swap(i, largest, arr)
      sort(arr, n, largest)
```

Thulathi Heap Sort :

```
ThulathiHeapSort( arr [0.........r])
n ← r
for i = (n/3 - 1)    to   0
     heapify(arr, n, i)
for i = n-1   to   0
    temp ← arr[0]
    arr[0]← arr[i]
    arr[i]← temp
    heapify( arr , n , 0)
```

Heapify:

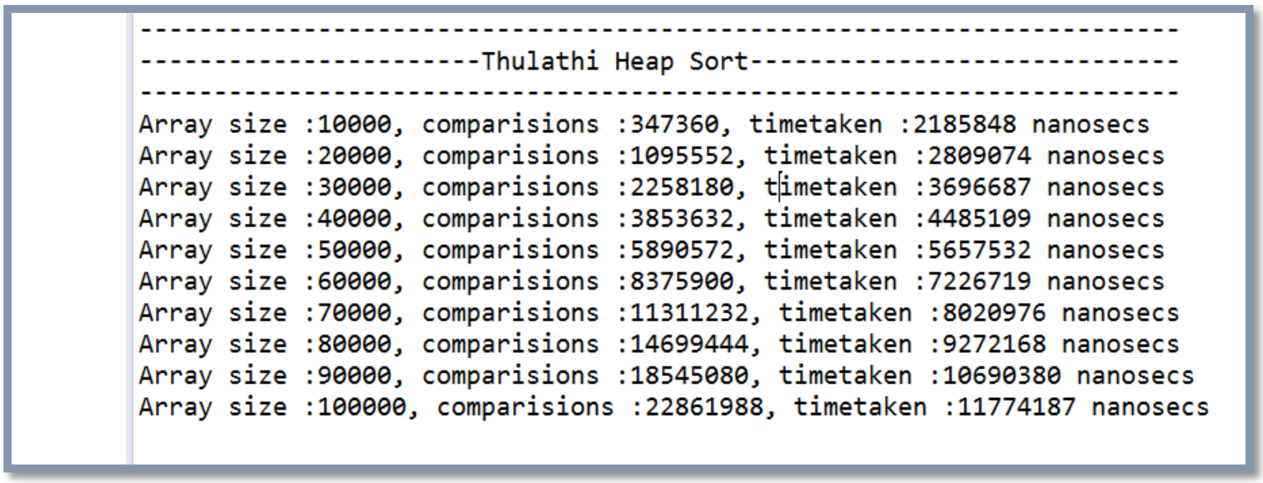
```
heapify( arr[0....r], n, i)
largest ← i
L ← ( 3 * i) + 1            //first child
mid ← ( 3 * i) + 2          //middle child
r ← ( 3 * i) + 3            //right child
if ( L != -1 and L < n and arr[L] > arr[largest] )
       largest = L
if ( mid < n and arr[mid] > arr[largest] )
       largest = mid
if ( r < n and arr[r] > arr[Largest] )
       largest = r
if ( largest != i)
       swap ← arr[i]
       arr[i] ← arr[largest]
      arr[largest] ← swap
      heapify(arr, n, largest)
```

Time complexity analysis for the algorithms :

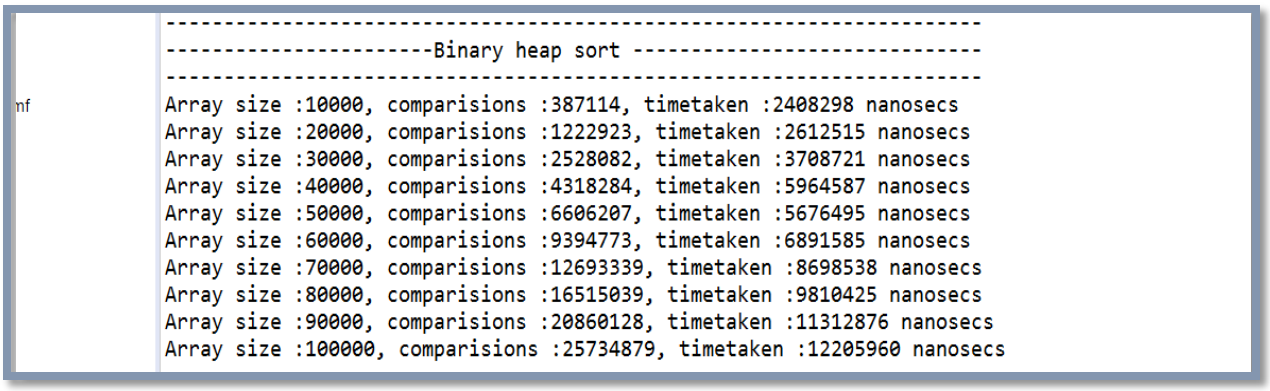
Binary Heap Sort = $n \log_2 nn$

Thulathi Heap Sort = $nn \log_3 nn$

screen shots:



```
----------------------------------------------------------------------
-----------------------Thulathi Heap Sort-----------------------------
----------------------------------------------------------------------
Array size :10000, comparisions :347360, timetaken :2185848 nanosecs
Array size :20000, comparisions :1095552, timetaken :2809074 nanosecs
Array size :30000, comparisions :2258180, timetaken :3696687 nanosecs
Array size :40000, comparisions :3853632, timetaken :4485109 nanosecs
Array size :50000, comparisions :5890572, timetaken :5657532 nanosecs
Array size :60000, comparisions :8375900, timetaken :7226719 nanosecs
Array size :70000, comparisions :11311232, timetaken :8020976 nanosecs
Array size :80000, comparisions :14699444, timetaken :9272168 nanosecs
Array size :90000, comparisions :18545080, timetaken :10690380 nanosecs
Array size :100000, comparisions :22861988, timetaken :11774187 nanosecs
```



```
----------------------------------------------------------------------
----------------------Binary heap sort -----------------------------
----------------------------------------------------------------------
Array size :10000, comparisions :387114, timetaken :2408298 nanosecs
Array size :20000, comparisions :1222923, timetaken :2612515 nanosecs
Array size :30000, comparisions :2528082, timetaken :3708721 nanosecs
Array size :40000, comparisions :4318284, timetaken :5964587 nanosecs
Array size :50000, comparisions :6606207, timetaken :5676495 nanosecs
Array size :60000, comparisions :9394773, timetaken :6891585 nanosecs
Array size :70000, comparisions :12693339, timetaken :8698538 nanosecs
Array size :80000, comparisions :16515039, timetaken :9810425 nanosecs
Array size :90000, comparisions :20860128, timetaken :11312876 nanosecs
Array size :100000, comparisions :25734879, timetaken :12205960 nanosecs
```