FNU ANIRUDH

Assignment-3

① We take one element of each list and put it in a min-heap. Along with each element we have to track which list we took it from.

While merging, we take the minimum element from the heap and insert another element off the list it came from (unless the list is empty). We continue until we empty the heap.

We have n steps and at each step we are doing an insertion into the heap, which is lg K.

Problem 6.2

(a) A d-ary heap can be implemented using a dimensional array as follows. The root is kept in A[1], its d children are kept in order in A[2] through A[d+1] and so on. The procedures to map a node with index i to its parent and its j th child are given by:

```
D-ARY PARENT(i)
  return ⌊(i-2)/d+1⌋

D-ARY CHILD(i,j)
  return d(i-1)+j+1
```

(b) Since each node has d children the height of the tree is $\Theta(\log_d n)$.

(c) The Heap-Extract-Max algorithm given in the text works fine for d-ary heaps; the problem is MAX-HEAPIFY. Here we need to compare the argument node to all its children. This takes $\Theta(d \log_d n)$ and dominates the overall time spent by HEAP-EXTRACT-MAX.

(d) The MAX-HEAP-INSERT given in the text works fine as well. The worst case running time is the height of the heap, that is $\Theta(\log_d n)$.

~~(e) The HEAP-INCREASE-KEY algorithm given in the text works fine.~~

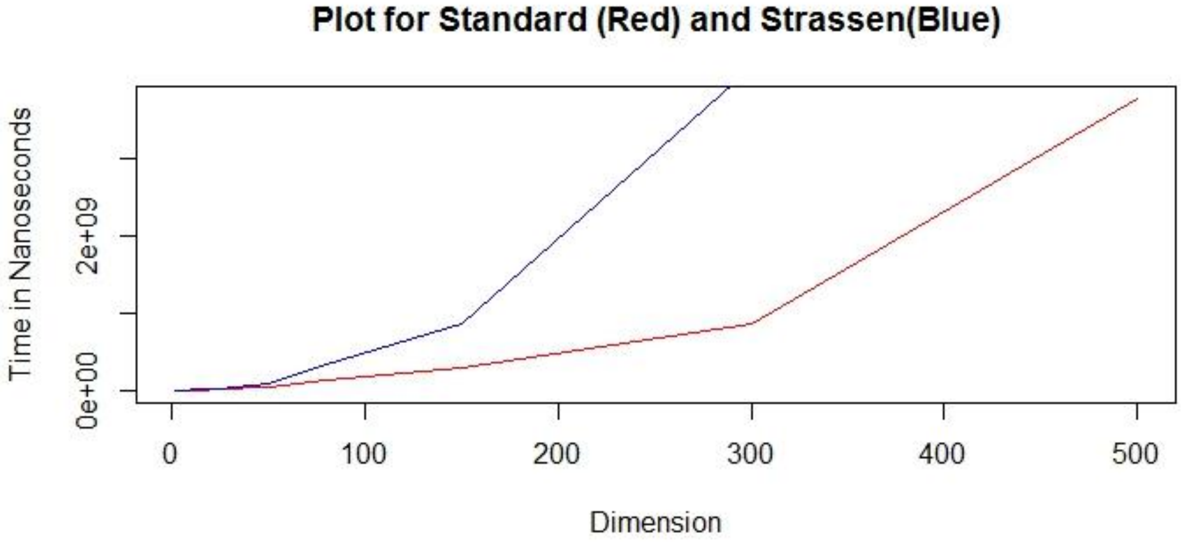
(e)
```
D-ARY-HEAP-INCREASE-KEY (A, i, k)
A[i] ← max(A[i], k)
while i > 1 and A[PARENT(i)] < A[i]
do exchange A[i] ↔ A[PARENT(i)]
i ← PARENT(i)
```

Problem 2 - 2:4

Please refer JAVA programs for Problem 2 to 2.4.

Problem 2.5

Plot for Time taken by Standard Matrix Multiplication and Strassen Algorithm in Nanoseconds



From Graph, We can say that Time taken for Standard and Strassen's Algorithm is same when dimension is close to 40.

- Strassen takes less time for higher dimension when compared to Standard matrix multiplication.
- Time is more or less similar when dimension is between 0 to 50.

Strassen's Algorithm can be improved by tuning i.e.:-

- Morton order that is based on a quad-tree decomposition of the matrix.
- select the recursion truncation point to minimize padding without affecting the performance of the algorithm.

Inefficiency of Strassen's Algorithm

- when we encounter matrices with one or more dimensions of odd size.
- Strassen's construction is no longer advantageous when recursion truncation point is reached.

List of References

1. http://www.sanfoundry.com/java-program-strassen-algorithm/

2. https://users.cs.duke.edu/~alvy/papers/sc98/