COSC262 Assignment 1 — Data Structures and Sorting

Problem 1. Prepare 10 random numbers of your choice, which require at least three rotations when inserted into an AVL tree. Trees can be hand-drawn if neatly done.

(1) Insert the above numbers into a binary search tree (BST). Show the tree after each insertion and the path from the root to the insertion point for each tree. [10 marks]

(2) Insert those numbers into an AVL tree. Show the tree after each insertion, and the tree after a rotation if a rotation is required. [10 marks]

(3) Delete the root twice with the above AVL tree. The AVL property must be kept. Balance labels (L, E, R) must be shown. [10 marks]

(4) Insert the above numbers into a 2-3 tree. Show the tree before and after each insertion. [10 marks]

Problem 2. Computer experiments.

(1) Implement heapsort, quicksort, mergesort and radix-10 sort in Python. Plot the computing time and number of comparisons for each method with inputs of random numbers. The size of each input data should change from 1000 to 20000. The results must be shown in a table and a graph. As a separate experiment try to do quicksort twice. You will need to stop with a small date size for this experiment. [10 marks]

(2) The next project is to design a radix sort with general radix r. Do experiment with several r with the same sets of data used in (1), and determine the best r. [10 marks]

(3) The next project is to design a hybrid of quicksort and radix sort, called quick-radix sort. Quick-radix sort is a sort of MSD (most significant digit) method. When we perform "partition" in quicksort, we inspect the leftmost bit in binary expansion. If it is 0, we put the key to the left list and if it is 1, we put it in the right list. The sorting process proceeds in a recursive fashion using the second most significant bit, and so on. Do experiment with the same data sets as in (1), and observe how much improvement can be gained or not. [10 marks]

(4) Discuss the observed performances in (1), (2) and (3) above. [30 marks]

Note. There are many sorting algorithms on the internet. You are not allowed to download any of them. You should stay within the course reader and this handout.

Note. The number of comparisons will be measured for heapsort, quicksort and merge sort.

Due date: April 11

Worth: 25%

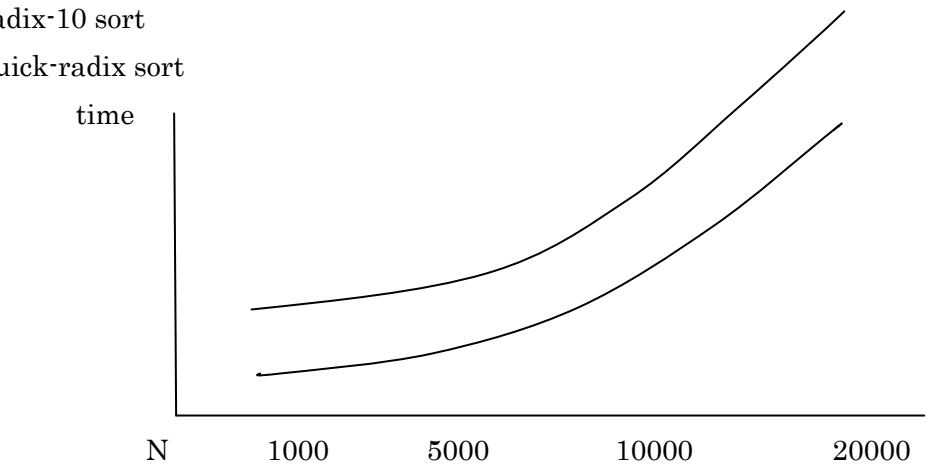
The table and graph for the results look as follows. Contents must be given from experiments. The result for the second quicksort can appear in the table for small n.

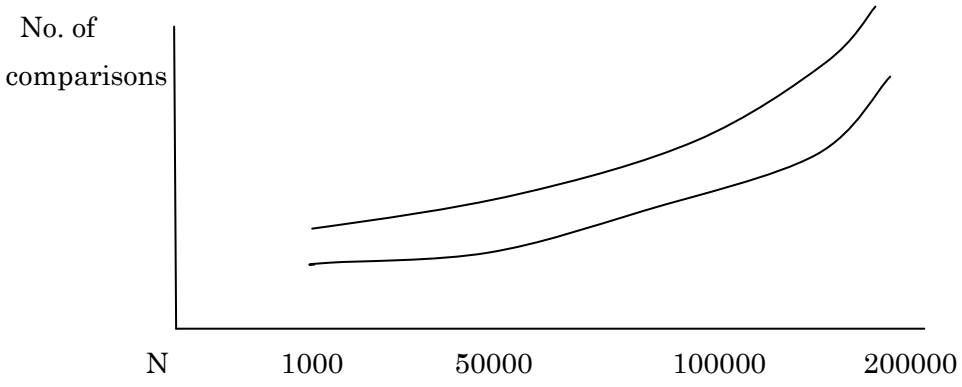
CPU times

n	1000	5000	10000	15000	20000
Heapsort					
Quicksort					
Merge sort					
Radix-10 sort					
Quick-radix sort					

Number of comparisons

n	1000	5000	10000	15000	20000
Heapsort					
Quicksort					
Mergesort					
radix-10 sort					
quick-radix sort					





Those curves are just imaginary. There will be more curves in the graph.

For radix-r sort, you need a separate table and graph.

```
# This is heapsort
# sort() sorts array a in descending order
# sift(p,q) heapifies array a from position p to q
# heap is a max-heap, that is, maximum at the root
import random
from time import time, clock
def out(n):
   for i in range(1, n+1): print a[i],
   print '\n'
def swap(i,j): # This swaps a[i] and a[j]
   w=a[i]; a[i]=a[j]; a[j]=w
def siftup(p, q):
    y=a[p]; j=p; k=2*p
    while k <= q
        z=a[k]
        if k < q :
           if z > a[k+1]:
              k=k+1
              z=a[k]
        if y ≤ z : break
        a[j]=z; j=k; k=2*j
    a[j]=y
def build_heap(n):
    for i in reversed(range(1,n/2+1)): siftup(i, n)
def sort():
   build_heap(n)
   for i in reversed(range(2,n+1)) :
      swap(1, i) # swap a[1] and a[i]
      siftup(1,i-1)
# {main program}
n=input('input n ')
a=[]
for i in range(0,10000): a=a+[int(100*random.random())]
out(n); t=clock()
sort(); out(n)
print 'time ',clock()-t
n=raw_input('finished ')
```

```
# This is quicksort
# sort(left,right) sorts array a from position left to right
# partition(left,right) partitions array a from position left to right
# with pivot x=a[left], and returns m such that after partition
# a[left .. m-1] <= x=a[m] <= a[m+1 .. right]
import random
from time import time, clock
def out(n):
    for i in range(1, n+1): print a[i],
    print '\n'
def sort(left,right):
    if left<right:
        m=partition(left,right)
        sort(left,m-1)
        sort(m+1,right)
def partition(left,right):
    x=a[left]; i=left; j=right+1
    while i<j:
        j=j-1
        if i==j: break
        while a[j]>=x:
            j=j-1; if i==j : break
        if i==j: break
        a[i]=a[j]
        i=i+1
        if i==j: break
        while a[i]<=x:
            i=i+1; if i==j : break
        if i==j: break
        a[j]=a[i]
    a[i]=x
    return i
# {main program}
n=input('input n ')
a=[]
for i in range(0,10000): a=a+[int(100*random.random())]
t=clock(); sort(1,n); out(n); print 'time ',clock()-t
n=raw_input('finished ')
```

```
# This is mergesort.    mergesort(p,q) sorts array a from position p to q
import random
from time import time, clock
c=0
def out(n):
   for i in range(1, n+1): print a[i], print '\n'
def mergesort(p, q):
   if p < q :
         m = (p+q) / 2;
         mergesort(p, m);
         mergesort(m+1, q);
         merge(p, m+1, q+1)
# merge(p,m,q) merges array a from position p to m and position m+1 to q
def merge(p, r, q):
   global c     ### c is comparison counter
   i=p; j=r; k=p;
   while i < r and j < q :
      c=c+1
      if a[i] <= a[j]:
         b[k] = a[i]; i=i+1
      else : b[k] = a[j]; j=j+1
      k=k+1
   while i < r :
      b[k]= a[i]; i=i+1; k=k+1
   while j < q :
      b[k] = a[j];
      j=j+1; k=k+1;
   for k in range(p,q): a[k] = b[k]
# main program
n=input('input n ')
a=[]
for i in range(0,20000): a=a+[int(100*random.random())]
b=[]
for i in range(0,20000): b=b+[0]
t=clock()
mergesort(1, n); out(n)
print 'time ',clock()-t, 'c=', c
n=raw_input('finished ')
```