CISC 5835 Algorithms for Big Data Fall, 2018

Homework Assignment #4

1 Answer the following questions about binary tree and heap:

(a) For a complete binary tree of height 4 (we consider a tree with just root node as having height 0), what's the range for the number of nodes in the tree? i.e., what's the maximum and minimum numbers of nodes for a complete binary tree of height 4, respectively? Draw both cases out below.

(b) If a heap has n elements, what's the height of the tree?

(c) If an array stores the following integers in the given order, is it a min-heap?

- Draw the heap, and label each node with the numbering of the nodes (i.e., root is node 1, root's left child is node 2, root's right child is node 3, ...).
- If a heap has n nodes, verify that nodes numbering from $n/2 + 1, ..., n$ are leaf nodes using this example. Also note that node i's parent is node $i/2$, and node i's left child is node $2i$, and its right child is node $2i + 1$.
- Check if the min-heap property is satisfied (How do you write a code to check this?)

```
9, 10, 12, 36 23, 11, 24, 21
```

(d) Use the heapq algorithm in Python to create a heap for the above data, and draw the resulting heap below:

(e) Illustrate the comparison and swapping happen involved when a smallest element is removed from the above heap.

(f) Illustrate the comparison and swapping happen involved when a new element 15 is inserted into the above heap after the above operation (of removing smallest element).

2 Exercises on data structures

(a) In dicrete-event simulation, events (with time-stamps and descriptions) are stored in a priority-queue, so that the main simulator loop will extract the next event from the queue (i.e., the one with the smallest timestamp) to process. In the space below, show how you would add a new event into the queue, and remove one event from the queue, using Python heapq algorithm. Hint: refer to the example in the handout.

```
## add event specified by a tuple (time, descrip) into the eventQueue
def AddNewEvent (eventQueue, time, descrip):
```

```
## remove and return the event in eventQueue that has the smallest timestamp
def NextEvent (eventQueue):
```

```
## test them
```

(b) Follow the outline in the handout, and use Python code to find out the results of the following set operations:

```
A={Red, Blue, Green, Yellow}
B={Circle, Square, Triangle, Rectangle}
C={a, b, c, d, e, f}
D={1, 3, 4, a, f, h}
```

```
## Write code to generate AXB (cartisian product of AxB, the set of all tuples with
## the first element taken from A, and second taken from B
```

```
# calculate the common elements from C and D
```

```
# calculate the union of C and D
```

(c) Demonstrate the insertion of the keys $3, 27, 18, 15, 20, 33, 12, 18, 9$ into a hash table with collision resolved by chaining. Let the table have 9 slots, and let the hash function be $h(k) = k \bmod 9$.

3 In the following pseudocode for iterative-mergesort (taken from the textbook), a FIFO queue is used to store the subarrays that are yet to be merged together.

```
function iterative-mergesort (a[1...n])
Input: elements a[1], a[2], ..., a[n] to be sorted

Q =  [] ## initialize Q to be empty queue

for i=1 to n:
    enqueue (Q, a[i...i]) ## insert a sublist, a[i] into queue

while the length of A is greater than 1:
     list1 = dequeue (Q)
     list2 = dequeue (Q)
     merged = merge (list1, list2) ## merged two sorted list into one    (##)
     enqueue (Q, merged)   (##)

return dequeue (Q) //in the end, return the single remaining list in the queue
```

(a) Suppose the input list to the function is as follows, show the content of Q after the while loop has been iteratied five times.

```
 ---------------------------------
| 6 | 1 | 4 | 2 | 3 | 8 | 7 | 5 |
 ---------------------------------
```

(b) Suppose that instead of queue, one uses stack to store the sublists to be merged. What would be the running time of the algorithm? Note: this means that we would replace enqueue() operation on queue by push() operation on stack (and dequeue() by pop()) operations.

4 For the wordcount example given below, a dictionary is used to keep track the number of occurrences of words in the input file. Expand the code so that it also reports the occurences of two-word sequences, such as (Expand, the), (the, code), (code, so), (so, that), ... are two-word sequences appear in this sentence.

```
import sys

filename = sys.argv[1] ## obtain file name from command line arguments

fp = open(filename) ## open the file

data = fp.read() ## read everything into data

words = data.split()  ## split data into words, using space, newline, tab as separator
fp.close()

unwanted_chars = ".,-_ (and so on)"
wordfreq = {} ## a dict, hashtable

for raw_word in words:
    word = raw_word.strip(unwanted_chars)
    if word not in wordfreq:
        wordfreq[word] = 0
    wordfreq[word] += 1


print(wordfreq)
```