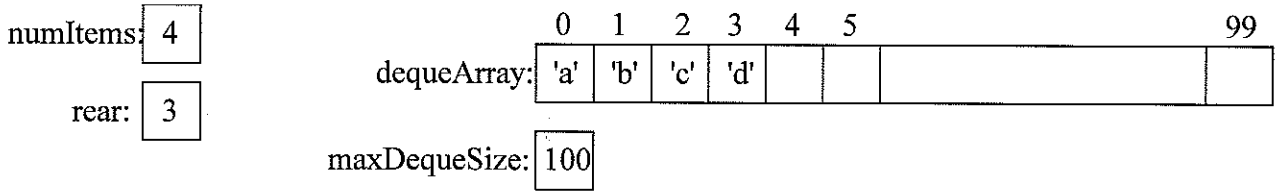
Spring 2010 Name: Mark Fienup

Data Structures - Test 2

Question 1. A Deque (pronounced "Deck") ADT is like a double-ended queue, i.e., it allows adding or removing items from either the front or the rear of the Deque.
One possible implementation of a Deque would be to use an array (dequeArray) to store the Deque items such that

- the front item is **always stored at index 0**,
- an integer numItems maintains the number of items in the Deq
- an integer rear maintains the index of the rear item
- an integer maxDequeSize maintains the size of the array



a) (12 points) Complete the worst-case theta notation, $\Theta(n)$, for each Deque operation, assuming the above implementation. Let n be the number of items in the Deque.

12

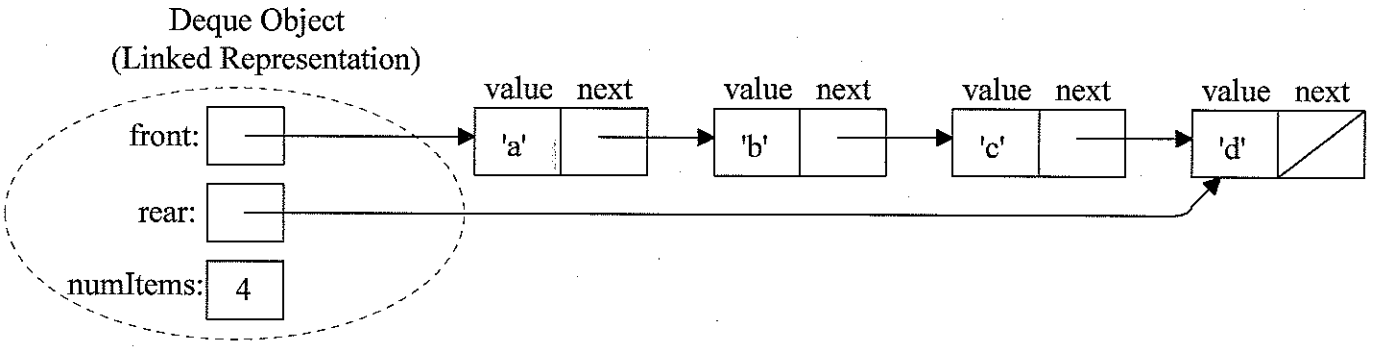
isEmpty	addFront	addRear	removeFront	removeRear	size
$\Theta(1)$	$\Theta(n)$	$\Theta(1)$	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$

b) (10 points) Suggest an improvement to the above array implementation of the Deque to speed up some of these operations.

10 Allow the front to "float" from other array-like treating array circularly.

— 22

Question 2. An alternative implementation of a Deque would be a linked implementation as in:



12

a) (12 points) Complete the worst-case theta notation, $\Theta(n)$, for each Deque operation assuming the above linked implementation. Let n be the number of items in the Deque.

isEmpty	addFront	addRear	removeFront	removeRear	size
$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	$\Theta(n)$	$\Theta(1)$

34

b) (16 points) Complete the removeRear method of the linked Deque implementation:

```
template <class T>
class DequeNode {
public:
      T value;              // Node value
      DequeNode<T> *next; // Pointer to next node

      // Constructor
      DequeNode (T nodeValue)
      { value = nodeValue;
      next = NULL;
}; // end class DequeNode


template <class T>
class Deque {
private:
   DequeNode<T> * front;    // Deque front pointer
   DequeNode<T> * rear;     // Deque rear pointer
   int numItems;            // Count of nodes

 public:
...};
```

+3 Special case: deleting last elt. => se
+4 loop to find node before rear
+3 delete node after getting value
+3 decrement numItems
+3 general ptr. manipulation

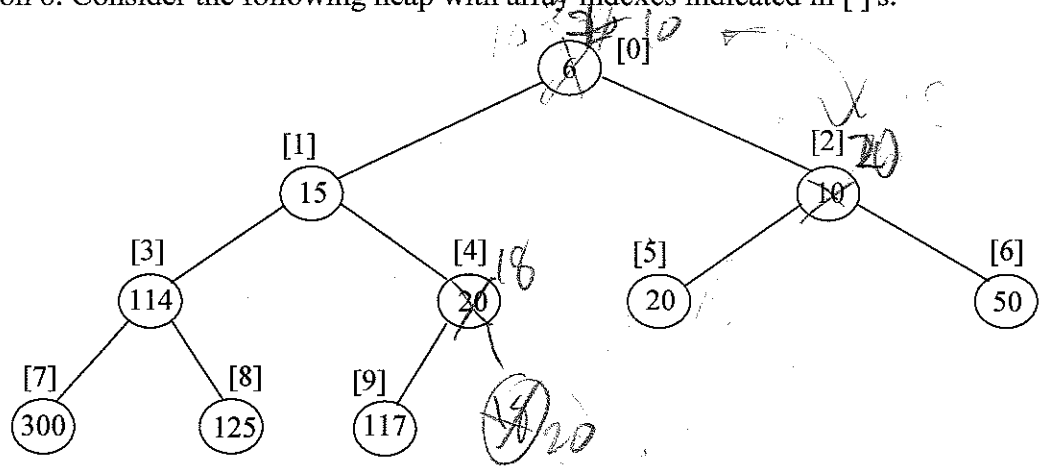
```
template <class T>
T Deque<T>::removeRear( ) {
   T value;
   if (rear == NULL) {
         assert("Cannot removeRear from empty Deque");
   } else if (rear == front) {
         value = rear->value;
         delete rear;
         rear = NULL;
         front = NULL;
   } else {
         value = rear->value;
         rear = front;
         while (rear->next->next != NULL) {
               rear = rear->next;
         }
         delete rear->next;
         rear->next = NULL;
   return value;
}
```

c) (5 points) Suggest a recommendation for improving the linked implementation of the Deque.

Doubly linked nodes so node before rear can be found easily.

5

Question 6. Consider the following heap with array indexes indicated in [ ]'s.



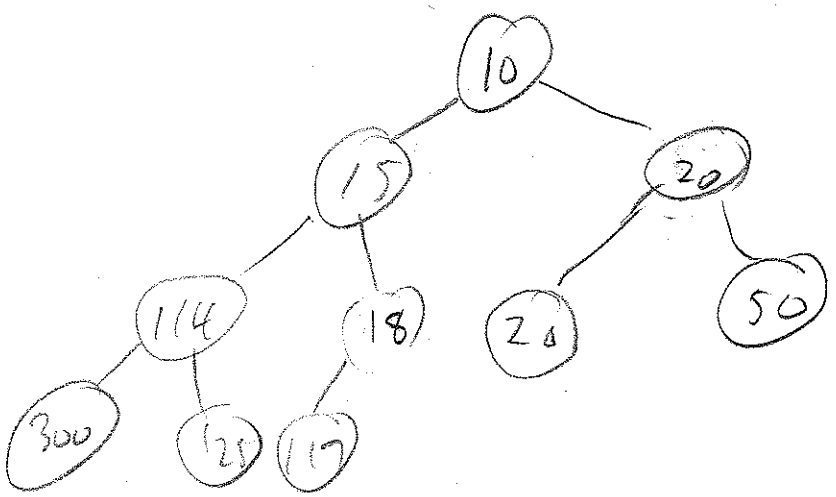
a) (4 points) For a node at index *i*, what is the index of:

4

- its left child if it exists: 2*i+1
- its parent if it exists: (i-1)/2

b) (11 points) What would the above heap look like after adding 18, **and then** dequeuing an item?

11



c) (10 points) Explain why adding a new item to a heap has a worst-case theta notation, $\Theta(\log_2 n)$, where n is the number of items in the heap

The height of a complete binary tree is $\log_2 n$ since the # of nodes on each level doubles

10

1
2
4
8
16

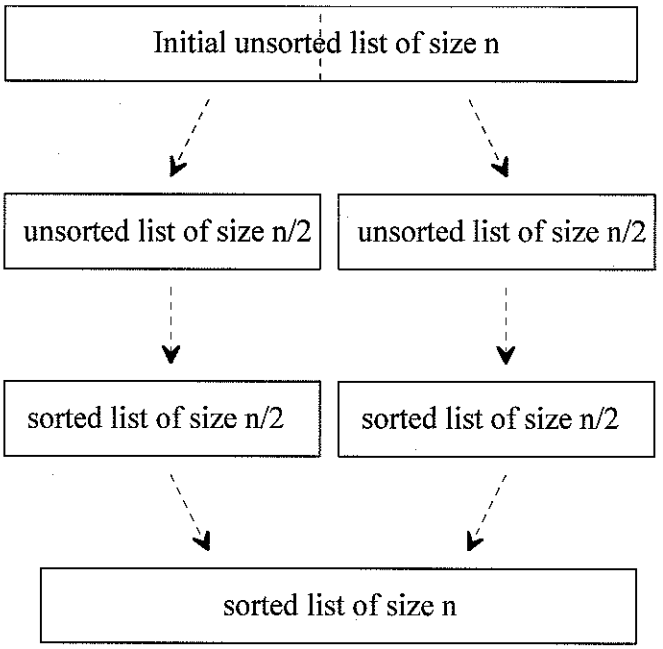
30

Question 7. Recall that merge sort is a recursive divide-and-conquer algorithm such that:

Divide - splits list/array into two equal parts

Conquer - recursively merge sort each half

Combine - merge the sorted halves back together



a) (5 points) When merging two sorted lists of size n/2 each, what is the worst-case number of comparisons that must be performed? (justify your answer for partial credit)

$(n-1)$ comparisons because in the worst case neither lists of size $\frac{n}{2}$ run out until we compare the last items.

5

b) (5 points) What maximum depth of recursion does the merge sort algorithm require when sorting a list of size n? (justify your answer for partial credit)

$\log_2 n$ since n is repeated cut in half to "arrays of size 1"

5

c) (10 points) Both Heap sort and Merge sort are $\Theta(n\log_2 n)$, where n is the number of items being sorted, but Merge sort takes about twice the time of Heap sort. Why?

The merge sort does more moves when copying to smaller arrays and merging back to larger array.

10

20