CPSC 221 Midterm 1

d6a2b

TOTAL POINTS

67.6 / 73

QUESTION 1

1 CSID 1 / 1

✓ **- 0 pts CSID Filled**

- 1 pts CSID Blank

QUESTION 2

Deque euqeD 6 pts

2.1 linked-list Omega(n) cost operation 0 / 2

- 0 pts Correct

- 0.5 pts popR or pushR only.

✓ **- 2 pts incorrect**

- 1 pts none or 1 correct + 1 incorrect

2.2 swap(p->next,p->prev); p=p->prev; 4 / 4

✓ **- 0 pts Correct**

- 0.5 pts swap is correct but p=p->next rather than p=p->prev

- 1.5 pts only one line is correct

- 2 pts only one line is partly correct (usually p=p->next)

- 4 pts incorrect/blank

- 0 pts Correct using push and pop

- 1 pts Correct using more than 2 lines

- 1.5 pts swap using more than 1 line and p=p->next rather than p=p->prev

- 2 pts Only swap correct using more the one line

QUESTION 3

Lines printed 12 pts

3.1 # Lines printed 4 / 4

✓ **- 0 pts <blank> D C A <blank> B (top to bottom) is correct**

- 1 pts 3 letters in correct boxes

- 2 pts 2 letters correct

- 3 pts 1 letter correct

- 4 pts none correct

3.2 Asymptotic 4 / 4

✓ **+ 4 pts D C A B (one per box, top to bottom) is correct**

+ 3 pts 3 letters in correct boxes

+ 2 pts 2 letters correct

+ 1 pts 1 letter correct

+ 0 pts none correct

3.3 Rank 4 / 4

✓ **- 0 pts A C B D (one per box, top to bottom) is correct or correct order for given asymptotic answers**

- 1 pts 3 letters in correct boxes or correct order, but bottom to top.

- 2 pts 2 letters correct

- 3 pts 1 letter correct

- 4 pts none correct

QUESTION 4

Tree traversal 10 pts

4.1 postorder 2 / 2

✓ **- 0 pts Correct (3, 2, 5, 6, 4, 1)**

- 2 pts Incorrect/Blank

4.2 different tree 2 / 2

- 0 pts Correct

- 2 pts Blank or incomplete

- 0 pts preorder=orig.postorder

- 0 pts postorder=orig.postorder

- 0 pts preorder=orig.postorder and orig.shape

- 0 pts preorder=orig.postorder and correct new shape

✓ **- 0 pts** **postorder=orig.postorder and gave new preorder**

- 2 pts same tree as orig.

- 1 pts close to correct postorder

- 0 pts postorder=orig.preorder

4.3 root, left-subtree, right-subtree 3 / 3

✓ **- 0 pts** **Correct**

- 3 pts blank/incorrect

- 1 pts incorrect underline left subtree

- 1 pts incorrect box right subtree

- 1 pts incorrect circled root

- 1 pts swapped left and right

4.4 full tree from postorder 3 / 3

✓ **- 0 pts** **Correct**

- 3 pts Blank/Incorrect

QUESTION 5

Weighty measures 11 pts

5.1 Number of weighings 2.7 / 3

- 0 pts Correct (1 1 2 2 2 2 2 2 3)

✓ **- 0.3 pts** **8/9 == 3**

- 1 pts correct up through 4 eggs

- 3 pts Incorrect

5.2 First comparison 3 / 4

- 0 pts Weigh $$\lfloor \frac{n+1}{3} \rfloor$$ against $$\lfloor \frac{n+1}{3} \rfloor$$

✓ **- 1 pts** **Weigh $$\lfloor \frac{n}{2} \rfloor$$ against $$\lfloor \frac{n}{2} \rfloor$$ (consider even AND odd case)**

- 4 pts Incorrect

- 0.5 pts Incorrect terms/floor

5.3 Fewest weighings 3.5 / 4

- 0 pts Correct $$ceil(log_3n)$$

✓ **- 0.5 pts** **Wrong Base (base 2 or other)**

- 4 pts Incorrect/Blank

- 0 pts Correct base log, but minor error (no ceil or other)

- 2 pts function that reduces space of problem (n-1), (n/2)

QUESTION 6

Quicksort 14 pts

6.1 Recurrence 3 / 3

✓ **- 0 pts** **Correct**

- 1 pts function of Lo and Hi but not using Q()

- 2 pts function of only n

- 3 pts incorrect/blank

- 2 pts recurrence assumes |Lo|=|Hi|

- 2 pts n-Lo and n-Hi with no additive cn

- 2 pts poorly remembered 3-way mergesort?

- 1 pts |Hi| and |Lo| are divided by 2

- 1 pts |Hi| and |Lo| are divided by n

- 1 pts includes Q(|Eq|)

6.2 Lo+Hi <= n-1 1 / 1

✓ **- 0 pts** **Correct $$n-1$$ or $$|A| - 1$$**

- 1 pts Incorrect

- 0.5 pts |A| or n

6.3 Empty Lo or Hi 3 / 3

✓ **- 0 pts** **Correct $$\theta(n^2)$$**

- 3 pts Incorrect

6.4 Lo, Hi <= 2n/3 3 / 3

✓ **- 0 pts** **Correct $$\Theta(nlogn)$$**

- 3 pts Incorrect

6.5 Possible pivots 2.5 / 3

✓ **- 0.5 pts** **missing one correct answer/contains one incorrect answer**

- 1 pts missing two correct answer/contains two incorrect answer

- 1.5 pts missing three correct answer/contains three incorrect answer

- 2 pts missing four correct answer/contains four incorrect answer

- 2.5 pts one correct answer

- 3 pts Incorrect/Blank

- **0 pts** Correct

6.6 Quicksort better than Mergesort 1 / 1

✓ **- 0 pts Correct**

- **1 pts** Incorrect

QUESTION 7

City planning 7 pts

7.1 B(5)=6 1 / 1

✓ **- 0 pts Correct (6)**

- **0 pts** Incorrect

7.2 Recurrence 3 / 3

✓ **- 0 pts Correct $$B(n-1)+n-2$$**

- **1.5 pts** First term correct, second incorrect
- **3 pts** Incorrect/Blank

7.3 Closed form 3 / 3

✓ **- 0 pts Correct**

- **3 pts** blank/incorrect
- **3 pts** a recurrence, a summation, or asymptotic notation
- **1 pts** close
- **2 pts** linear or n\log n or cubic or exponential or factorial

QUESTION 8

Big sum 12 pts

8.1 bestAt 3 / 3

✓ **- 0 pts Correct**

- **0.25 pts** A[0] = 0, or extremely simple addition error, otherwise correct.
- **0.5 pts** solution went off rails at pos 6, 7, or 8
- **1 pts** Solution went off rails at pos 4 or 5.
- **2 pts** mis-interpretation of bestAt, but some evidence of cumulative sums.
- **3 pts** blank or shows no accumulation of elements of A.

8.2 Ending at position 1 / 1

✓ **- 0 pts Correct $$i - 1$$; ($$A[i-1]$$ not accepted)**

- **1 pts** Incorrect

8.3 best 1 / 1

✓ **- 0 pts Correct**

- **0.5 pts** general off-by-one on indices, or no mention of A.
- **1 pts** blank, bestAt[i], A, bestAt[i-1], A[0..n-1], i-1, A[i], all other incorrect.

8.4 Base case true? 1 / 1

✓ **- 0 pts Correct "No"**

- **1 pts** Incorrect/Blank

8.5 Base case fix 0.9 / 1

- **0 pts** Correct

✓ **- 0.1 pts best = A[0]**

- **1 pts** Blank/Incorrect

8.6 Key fact: position i-1 0 / 1

- **0 pts** Correct $$i-1$$; $$A[i-1]$$ not accepted

✓ **- 1 pts Incorrect/Blank**

8.7 Key fact: just A[i] 1 / 1

✓ **- 0 pts A[i]**

- **1 pts** incorrect/ blank

8.8 Inductive step (A): bestAt[i]=bestAt[i-1]+A[i] 1 / 1

✓ **- 0 pts A[i]**

- **1 pts** incorrect/ blank

8.9 Inductive step (A): bestAt[i]=A[i] 1 / 1

✓ **- 0 pts bestAt[i-1]+A[i]**

- **1 pts** incorrect/ blank

8.10 Inductive step (B): 11 1 / 1

✓ **- 0 pts Correct (11)**

- **1 pts** Incorrect

CPSC 221 2019W1: Midterm Exam 1

October 10, 2019

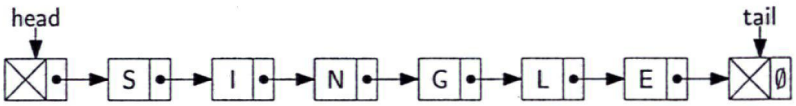
1 Who gets the marks? [1 mark]

Write your 4 or 5 digit CSID here: d6926

2 Deque euqeD [6-marks]

A double ended queue or, *deque* for short, is a queue that allows push and pop operations to be performed from both ends of the queue. The four operations `pushL`, `popL`, `pushR`, and `popR` push and pop from the left and right end of the queue, respectively. In the problems below, please assume that the data element in a node **cannot** be changed.

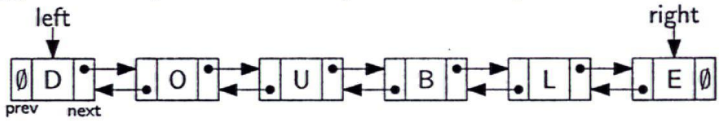
(a) **[2-marks]** Suppose we use a singly-linked list with a dummy head and a dummy tail to implement a deque. The head of the list is at the left of the deque and the tail is at the right.



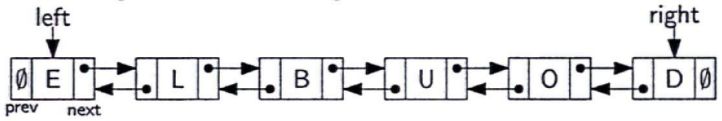
Which operations must take $\Omega(n)$ time in this implementation of a deque containing n items (assuming you may use only a constant amount of additional space).

⊗ `pushL` ⊗ `popL` ○ `pushR` ○ `popR` ○ none

(b) **[4-marks]** Now suppose we implement the deque as a doubly-linked list with no dummy nodes.



Add two lines to complete the following code to reverse the contents of the deque.



You may use the function `swap` which **only** swaps two `Node` pointers (which are passed by reference).

```
struct Node { string data; Node *prev, *next; };
void DLL::reverse() {
  Node *p=left;
  while( p != NULL ) {
```

swap(p->prev, p->next);

p = p->prev;

```
  }
  swap(left,right);
}
```

3 Lines printed [12-marks]

Enter the letter of the code fragment next to the number of lines printed by the fragment as a function of n, this function in asymptotic notation, and the rank of this function compared to the other fragments (1=smallest). Some boxes will be empty and some boxes might contain two letters. No box in the # Lines printed column contains two letters. You may assume n is a power of 2.

$2^k = n$

Code fragment	# Lines printed	Asymptotic	Rank

A
`for(int i=1; i<n*n; i=i*2)`
`  cout << "A" << endl;`
		$\log_2(n)$
		$k \to 2^k = n$
			D	$\Theta(n^2)$	A	1

B
`for(int i=1; i<n; i=i*2)`
`  for(int j=1; j<=i; j++)`
`  cout << "B" << endl;`
	D	~~$n(n+1)/2$~~
		$n(n-1)/2$
			C	$\Theta(\sqrt{n})$	C	2

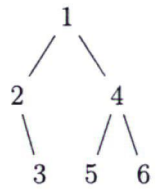
C
`int i=1; int s=1;`
`while(s <= n) {`
`  cout << "C" << endl;`
`  s = s + 2*i + 1;`
`  i = i + 1;`
`}`
	C	$\lfloor\sqrt{n}\rfloor$	A	$\Theta(\log n)$	B	3

D
`for(int i=0; i<n-1; i++)`
`  for(int j=i+1; j<n; j++)`
`  cout << "D" << endl;`
	A	$\log_2(n^2)$	B	$\Theta(n)$	D	4

		n^2				
	B	$n-1$				

4 Tree traversal [10-marks]

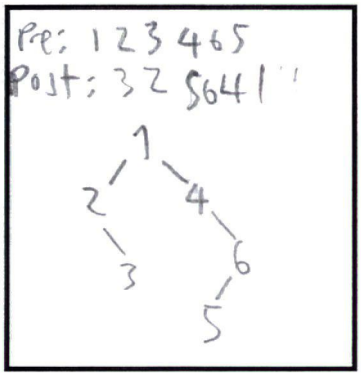
For this question, you may assume that all trees are **binary** trees and that their nodes have been labeled from 1 to n in the order of a **preorder** traversal. For example, one such labeled tree is:



(a) **[2-marks]** What is the sequence of labels produced by a postorder traversal of the above tree?

3 2 5 6 4 1

(b) **[2-marks]** Draw a **different** binary tree, with its preorder labeling, that produces the same sequence of labels by a postorder traversal as the above tree.



(c) **[6-marks]** This sequence of labels was produced by a postorder traversal of a **full** binary tree whose **preorder** traversal was labeled from 1 to 11 (in order).

not sorted [0]

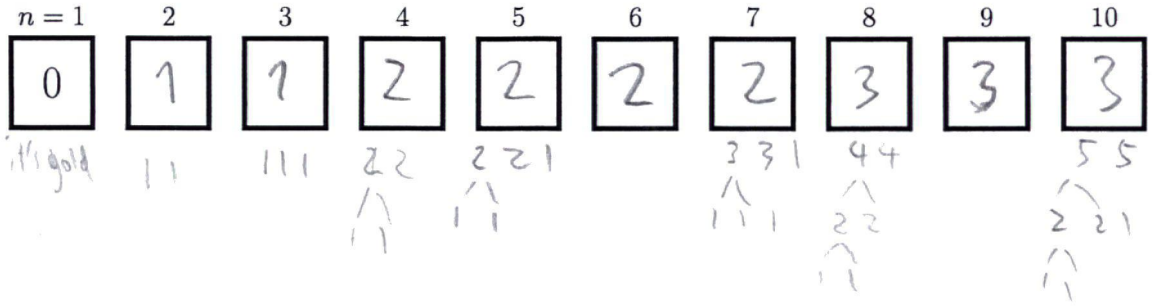
3 5 6 4 2 9 10 8 11 7 1

Circle the label of the root in the sequence, underline the labels in the left subtree of the root, and draw a box around the labels in the right subtree of the root. Finally, use the recursive definition of a full binary tree to reconstruct the original tree,

5 Weighty measures [11-marks]

You have been given a prize! It's a basket of n eggs. One of the eggs contains a gold coin. The other eggs are rotten, and you do **not** want to open one of those. Fortunately, the gold egg weighs slightly more than a rotten egg (all rotten eggs weigh the same) *and* you have a balance that can compare two piles of eggs. The balance will tell you if one pile of eggs weighs **less** or **more** or **the same** as the other pile. You should assume that the only way to gain information about eggs is to weigh them against other eggs. So no metal detectors or x-ray machines or anything like that.

(a) **[3-marks]** Fill in the following table with the fewest number of weighings in the worst case needed to find the gold egg among n eggs for $n = 2, \ldots, 10$.



(b) **[4-marks]** What is the **first** comparison one should perform to find the gold egg among n eggs using the fewest number of weighings in the worst case?

Divide eggs into two equal pile. If even, heavier pile has golden egg, and other half discarded. If odd, heavier pile has golden egg, and other half discarded, but if 2 pile same weight, then the one egg in neither pile is gold.

(c) **[4-marks]** What is the fewest number of weighings, as a function of n, that the best algorithm needs in the worst case to find the gold egg? Your solution should not be a recurrence, contain a summation, or use asymptotic notation. (Note: This is a question about the complexity of solving a problem using a particular model of computation.)

$\lfloor \log_2(n) \rfloor = O(\log n)$

6 Quicksort [14-marks]

Quicksort is a recursive sort like mergesort. It chooses one element p, called *the pivot*, from its input array A and splits the input into those elements that are Lower, Equal, and Higher than the pivot. Then, it recursively sorts the Lower elements and recursively sorts the Higher elements. Finally, it concatenates (using $\circ$) the sorted Lower, the Equal, and the sorted Higher elements to produce its output.

Here's an outline of how it works:

1. **Quicksort**$(A[0 \ldots n-1])$
2. if $(n \leq 1)$ return A 1
3. $p =$ **pivot**(A) cn
4. $Lo =$ elements in A that are $< p$ cn
5. $Eq =$ elements in A that are $= p$ (including p) cn
6. $Hi =$ elements in A that are $> p$ cn
7. $Lo =$ **Quicksort**(Lo) Q
8. $Hi =$ **Quicksort**(Hi) Q
9. return $Lo \circ Eq \circ Hi$ cn

(a) **[3-marks]** Let $Q(n)$ be the worst-case number of steps taken by **Quicksort** on an input of size n. Assume that lines 3,4,5,6, and 9 take a total of cn steps. What is the recurrence equation for $Q(n)$ expressed using functions of c, n, $|Lo|$, and $|Hi|$ (where $|X|$ is the size of array X)?

$$Q(0) = Q(1) = 1$$

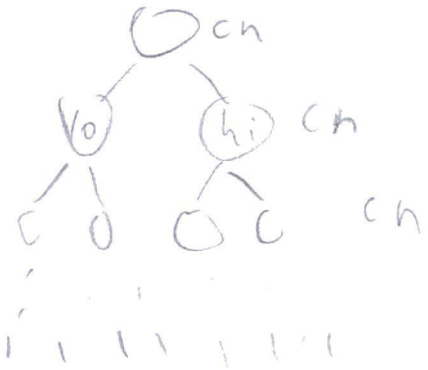
$$Q(n) = \boxed{1 + cn + Q(|Lo|) + Q(|Hi|)} \quad \text{for } n > 1$$

(b) **[1-marks]** The function **pivot**(A) returns an element from the array A. This implies that

$$|Lo| + |Hi| \leq \boxed{n - 1}$$ $1 + |Lo| + |Hi| = n$

(c) **[3-marks]** Recall that we defined $Q(n)$ to be the worst-case number of steps taken by **Quicksort** on an input of size n. Suppose **pivot** always returns a pivot p so that one of Lo and Hi is empty. What is $Q(n)$ in this case?

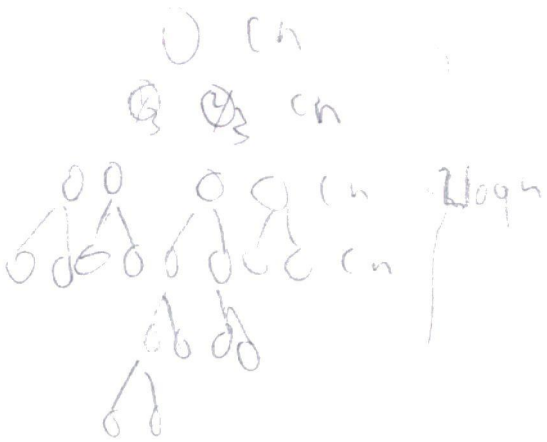
○ $\Theta(n)$ ⊗ $\Theta(n^2)$ ○ $\Theta(\log n)$ ○ $\Theta(n^2 \log n)$ ○ $\Theta(n \log n)$



(d) **[3-marks]** Suppose function **pivot** always returns a pivot p so that both $|Hi|$ and $|Lo|$ are at most $2n/3$. What is $Q(n)$ in this case?

~~○ $\Theta(n)$~~ ~~○ $\Theta(n^2)$~~ ○ $\Theta(\log n)$ ○ $\Theta(n^2 \log n)$ ⊘ $\Theta(n \log n)$

$Q(n) = n + Q(\frac{n}{3}) + Q(\frac{n2}{3})$



(e) **[3-marks]** Suppose we remove lines 7 and 8 from **Quicksort** so it makes no recursive calls but only splits the input into Lo, Eq, and Hi and then returns $Lo \circ Eq \circ Hi$. What are the possible values for the pivot p if the returned array is:

10, 7, 5, 12, 19, 13, 16, 21, 22, 39, 35, 28, 37, 46, 56, 61, 54, 73, 86, 75, 99

Possible pivots: 12 , 21, 22, 46, 61, 73, 99

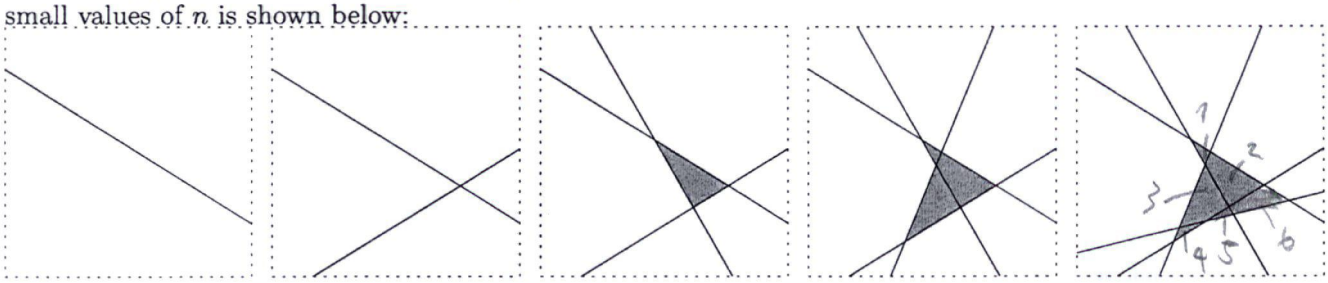
(f) **[1-marks]** Is **Quicksort** asymptotically faster than Mergesort? ○ Yes ⊘ No

nlogn

7 City planning [7-marks]

We've hired a team of engineers to build streets for a new city. The amazing part is that every street is perfectly straight and **every pair of streets intersects**! The streets divide the city into regions. Some regions are completely surrounded by streets (they are *bounded*) and others are not. If the engineers build n streets, how many bounded regions are there?

Let $B(n)$ be the number bounded regions in such an arrangement of streets. The value of $B(n)$ for small values of n is shown below:



$B(1) = 0$ $B(2) = 0$ $B(3) = 1$ $B(4) = 3$ $B(5) =$ 6

1 2 3

(a) **[1-marks]** Fill in the value of $B(5)$. You should trace the dashed portion of the street to see exactly how bounded regions are created.

(b) **[3-marks]** Complete the following recurrence relation for $B(n)$:

$$B(1) = 0$$

$$B(n) = B(\boxed{n-1}) + \boxed{n-2} \quad \text{for } n \geq 2$$

(c) **[3-marks]** What is $B(n)$ as a function of n? Your solution should not be a recurrence, contain a summation, or use asymptotic notation. (To check your formula, note that $B(6) = 10$.)

$$B(n) = \frac{(n-2)(n-1)}{2}$$

$\frac{n(n+1)}{2} \to \frac{(n-2)(n-2+1)}{2} : \frac{(n-2)(n-1)}{2}$

$\frac{4(5)}{2} = \frac{20}{2} = 10$

pretty sure this was in one of CS121's past summer session's exams

8 Big sum [12-marks]

The following code is intended to find the largest sum of contiguous elements (elements that occupy consecutive locations) in an input vector of positive and negative integers. But you're not sure it works correctly.

```
1  int bigSum(vector<int> & A) {
2    vector<int> bestAt(A.size());
3    bestAt[0]=A[0];
4    int best=0;
5    for( int i=1; i<A.size(); i++ ) {
6      if( bestAt[i-1] < 0 ) {
7        bestAt[i] = A[i];
8      } else {
9        bestAt[i] = bestAt[i-1]+A[i];
10     }
11     if( best < bestAt[i] ) best = bestAt[i];
12   }
13   return best;
14 }
```

(a) **[3-marks]** Show the contents of the `bestAt` vector at the end of the call to `bigSum` on the following input array `A`:

	0	1	2	3	4	5	6	7	8
A	-3	1	3	-2	4	6	-4	5	-1
bestAt	-3	1	4	2	6	12	8	13	12

(b) **[2-marks]** Fill in the blanks for the following loop invariant:

For all $i \in \{1, 2, \ldots, n\}$, where $n =$ `A.size()`, at the start of iteration i,

(A) `bestAt[i-1]` is the largest sum of contiguous elements in `A` ending at position [i − 1]

(B) `best` is the largest sum of contiguous elements in [A[0] ... A[i-1]]

(c) **[2-marks]** Is the base case true? ○ Yes ⊗ No

If not, explain how to modify the code to make it true:

```
int best = best+A+[0];
```

(d) **[2-marks]** Fill in the blanks of the following **key fact** about sums of contiguous sequences ending at position i.

The contiguous sequence with largest sum ending at position i is either:

1) the contiguous sequence with largest sum ending at position [A[i-1]] followed by $A[i]$, or

2) just [A[i]]

(e) **[2-marks]** Suppose that at the start of iteration i the loop invariant (from part (b)) is true (inductive hypothesis). We want to show that at the start of iteration $i+1$ (i.e., the end of iteration i) the loop invariant is true. Fill in the blanks below to explain how to use the key fact to show that invariant (A) holds for iteration $i+1$.

If `bestAt[i-1] < 0` then the inductive hypothesis and the key fact imply that `bestAt[i] =`

[A[i]].

Otherwise, the inductive hypothesis and the key fact imply that `bestAt[i] =`

[bestAt[i-1]+A[i]].

(f) **[1-marks]** What line of code ensures that invariant (B) hold for iteration $i+1$? [11]

This page intentionally left (almost) blank.
If you write answers here, you must CLEARLY indicate on this page what question they belong with AND on the problem's page that you have answers here.

This page intentionally left (almost) blank.

If you write answers here, you must CLEARLY indicate on this page what question they belong with AND on the problem's page that you have answers here.

CS 221 Poem:

Data structure by [illegible]nda and Will
They might grill us all [illegible]
If we don't try to study [illegible]
Our report cards may never [illegible]
But both are pretty [illegible]
TAs are [illegible] and none of them [illegible]
Everyone's mission is the same
It's a game of getting all student [illegible]