

The test will be open book, open notes, 75 minute time limit. Please write your answers on the exam sheet. Nine problems, 10 points per problem. No electronic devices allowed (except a music player that nobody else can hear, and whose buttons you don't push during the exam).

1. *Analysis of non-recursive algorithms.* Consider the problem of finding the longest palindrome occurring as a substring of a given input string s . For example, if we are given *abaracecardog* we should find *racecar*. (There might, of course, be two palindromes of equal longest length. In that case, we want the one starting nearest the beginning of the input string.) Here *palindrome* means *simple palindrome*, that is, a string that is its own reverse (unlike in the review exercise where punctuation and spaces were ignored). You will not be asked to write code to solve this problem, but rather, to analyze two possible approaches to the problem. Let N be the length of the input string.

Plan A: First write a method *Palindrome*(String z) that checks if its input is or is not a palindrome. Then, to solve the main problem, examine, for each i and j with $i+j < N$, the substring of j characters beginning at index i . Call *Palindrome* to check if that substring is a palindrome and keep track of the longest one found so far.

Plan B: examine, for each $i < N$, the longest palindrome centered either at the i -th input character or (in the case of even-length palindromes) between the i -th input character and the next input character, by looking for the first mismatch between characters at $i-k$ and $i+k$ (for odd-length palindromes) or between characters at $i-k$ and $i+k+1$ (for even-length palindromes).

(a) Which plan leads to the fastest code, plan A or plan B? **Plan B**

(b) Give the running time of plan A in terms of N using Θ -notation. $\Theta(N^3)$

*Explanation: it takes three nested loops, one over i , one over j , and one in *Palindrome*.*

(c) Give the running time of plan B in terms of N using Θ -notation. $\Theta(N^2)$

Explanation: it takes two nested loops, one over i and one over k .

(d) What is the largest value of N for which you would expect to be able to run your chosen program in one minute on a 1 gigahertz machine? $10^9 N^2 = 3600$ seconds times 10^9 operations per second = 3.6×10^{12} , so N is 600,000 or so. The 10 on the left is for the loop body; if it's 20 instead we get 400,000 instead of 600,000.

2. Indicate, for each pair of expressions (A,B) in the table below, whether A is O of B. Write “Yes” or “No” in each blank box of the table.

A	B	is A = O(B)?
n^{27}	2^n	yes
$\lg n$	n	yes
$\lg n + \sqrt{n}$	n	yes
\sqrt{n}	$\sqrt{n} + 5$	yes
$\sqrt{n} + 5$	\sqrt{n}	yes
$n \lg n$	n^2	yes
$\lg n$	\sqrt{n}	yes

3. Use Θ -notation to express the worst-case running times of the following algorithms that have been discussed in class:

(a) naive sorting an array of length n using this code:

```
for(i=0;i<n;i++) for(j=i+1;j<=n;j++)
if(x[j] < x[i]) swap(x[i],x[j]);
```

Answer: $\Theta(n^2)$

(b) A faster sorting algorithm, on an array of length n . Answer: $\Theta(n \lg n)$

(c) Binary search, on a sorted array of length n . Answer: $\Theta(\lg n)$

(d) Testing an n -digit integer M to see if it is a prime, by dividing it in turn by each number up to \sqrt{M} .

Answer: $\Theta(2^{n/2})$ **Note: the number M itself is about 2^n , where n is the number of digits; so its square root is about $2^{n/2}$.**

(e) Multiplying two n by n matrices, when a matrix is represented as `double[][]`.

Answer: $\Theta(n^3)$ **Explanation: this takes three nested loops, one over rows of the first matrix, one over columns of the second matrix, and one more to compute the entry at a given row and column.**

4. *Linked lists*. The class *StringList* has these members:

```
{ String key;  
  StringList next;  
}
```

Write a static method *void replace(String target, String replacement, StringList x)* that replaces every occurrence of *target* in the list *x* changed to *replacement*. (Not just the first occurrence.) The input *null* for *x* is legal, meaning *x* is an empty list. No list nodes are allocated; only change the data in the existing nodes. For example, if *x* is (*cat, dog, giraffe, dog, deer*) and we call *replace("dog", "ant",x)*, then afterwards *x* is (*cat, ant, giraffe, ant, deer*).

```
public static void replace(String target,  
                           String replacement, StringList x)  
{ StringList marker;  
  for(marker = x; marker != null; marker=marker.next)  
    { if(marker.key.equals(target))  
      marker.key = replacement;  
    }  
}
```

5. *Sparse matrices.* Consider an n by n matrix M that has 1 in the diagonal entries and in the entries next to (above or below) the diagonal, i.e. in the (i,j) positions where j is equal to i or $i + 1$ or $i-1$, and M has zeroes everywhere else. For example, when $n = 5$, M looks like this:

```

1  1  0  0  0
1  1  1  0  0
0  1  1  1  0
0  0  1  1  1
0  0  0  1  1

```

(a) Draw a linked-list diagram showing the representation of this particular matrix (for $n=5$) in the *BigMatrix* representation.

Answer: I don't have a good way to produce linked-list diagrams in electronic form, so this is the best I can do:

```

rows[0] --> (0,1) --> (1,1)
rows[1] --> (0,1) --> (1,1) --> (2,1)
rows[2] --> (1,1) --> (2,1) --> (3,1)
rows[3] --> (2,1) --> (3,1) --> (4,1)
rows[4] --> (3,1) --> (4,1)

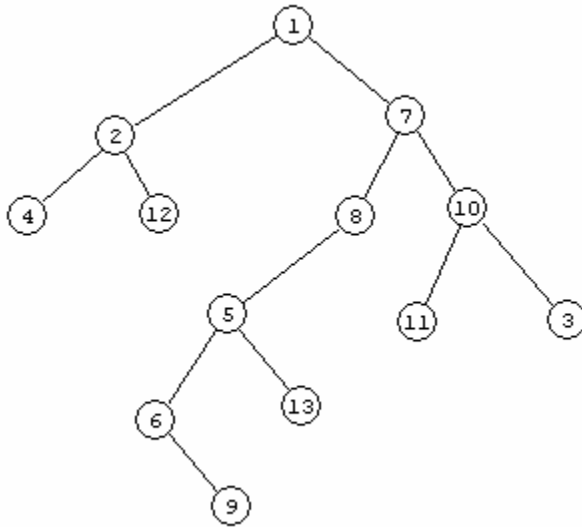
```

(b) Use Θ -notation to express the number of bytes required to represent the matrix M as a function of n .

Answer $\Theta(n)$.

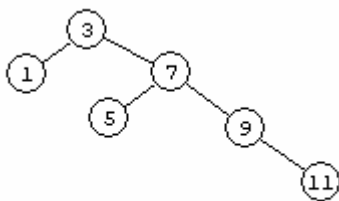
6. Binary Search Trees.

The picture represents a binary search tree. **The numbers shown are arbitrary node labels, not numbers representing the contents of the nodes. The contents are not shown.** If node 1 is deleted, using the textbook algorithm for binary search tree deletion, what will be the new root node? *Answer: node 6, which is the successor of 1.*

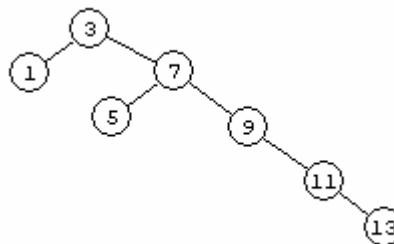


7. Red-black trees. For each of the following two trees, either indicate how to color the nodes red and black (use R and B to label the nodes if you don't have a red pen or pencil) to make the tree a red-black tree, or explain why that is not possible. In these pictures, the NIL leaf nodes are not shown.

(a)



(b)

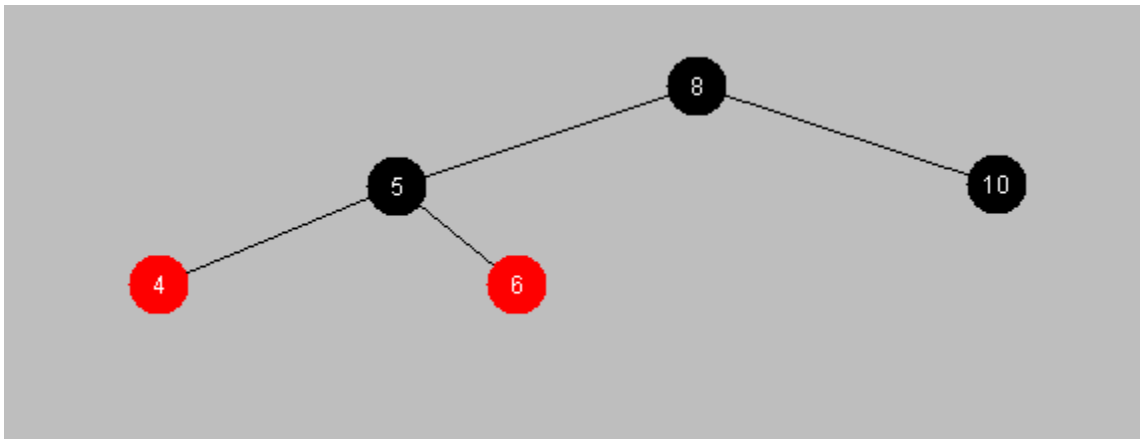


Answer for (a): Make 3, 1, 4, and 9 black and 7, 11 red.

Answer for (b): it is not possible. We would have to make 3 and 1 black, so the black height of 3 would be 2, and then to achieve as low a black height on the right as possible we would still need to make at least 9 and 13 black, which is not consistent with 3 having black height 2.

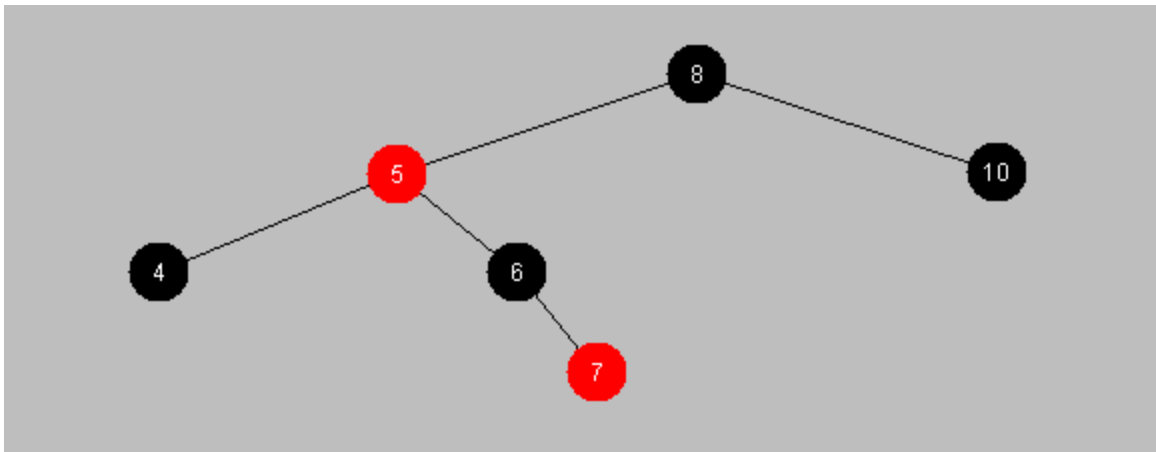
8(a). *Red-black insertion.*

Node 7 is to be inserted into the following red-black tree. In these pictures, the NIL leaf nodes are not shown.



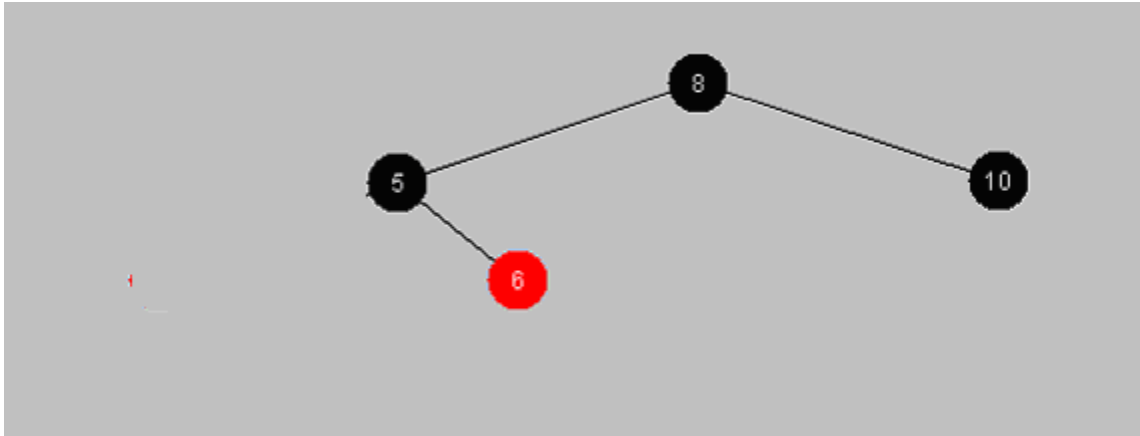
Draw the tree after this insertion.

Answer: This is the case when 7 has a red uncle. So we just leave the tree structure alone, and change the colors of the parent, uncle, and grandparent of 7:



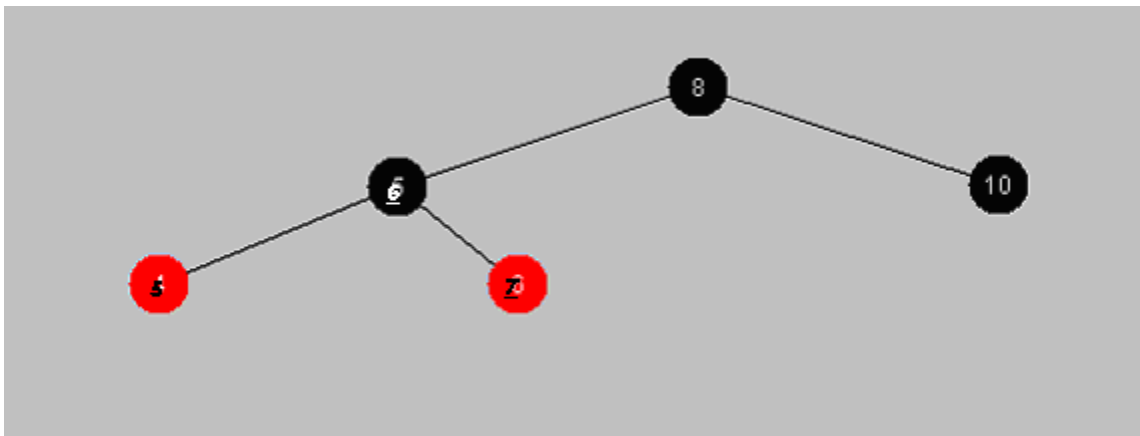
8(b). *Red-black insertion.*

Node 7 is to be inserted into the following red-black tree. In these pictures, the NIL leaf nodes are not shown.



Draw the tree after this insertion.

Answer: Now 7 has a black uncle (the invisible NIL, left child of 5). Since 7 is more than 6, it's a straight line from 7 to its grandparent, so we're in Case 3 of the algorithm and we need to left-rotate 5 and 6.



(Sorry for the not-so-beautiful appearance of the diagram.)