This prelim has 4 questions. Be sure to answer them all. Please write clearly, and show all your work. It is difficult to give partial credit if all we see is a wrong answer. Also, be sure to place suitable comments --method specifications, variable definitions-- in your programs.

Write your name and netid at the top of each page.

**Question 1 Sorting** (25 points).

**Part (a)** Write an algorithm to sort array segment b[0..n-1], where n\(\geq 0\).

Your algorithm must be a selection sort. We expect you to write
(a) the precondition
(b) the postcondition
(c) the invariant (perhaps as a diagram or picture)
(d) the bound function

and then to write the loop. Do not write a complete method.

Just write the loop with initialization. You don’t have to write an inner loop, if you write the body at a suitable level of abstraction.

**Part (b)** Give the worst-case order-of-execution time and best-case order-of-execution time of mergesort and of quicksort.

**Part (c)** Give the worst-case and best-case times of the binary search that we wrote in class (and that appears in the handout on correctness) --Given x and b[0..n-1], it looks for an index k that satisfies b[k] \(\leq x < b[k+1]\).
Question 2. Loops and linked lists (25 points). Consider a (singly-)linked list b without a header. Its nodes are of class ListNode. ListNode has the usual field next. In the diagrams, the second component of each folder is field next.

Write a loop with initialization that reverses the linked list by changing b and field next of each node. The linked list may be empty, in which case b contains null — in the diagrams, null is represented by ⊥⊥⊥⊥. Your algorithm shouldn’t use any variables except b and c (but it can declare variables within the loop body). The notations an, ah, aj, ak, etc. cannot appear in your algorithm; they are used only to describe the problem. Your grade depends solely on our being able to check the correctness of your work using the check list for understanding loops — so you should use it in developing the algorithm. It doesn’t matter how well your algorithm “works” if it does not use our invariant and bound function.

Precondition Q: The linked list looks like: 

Postcondition R: The linked list looks like: 

Bound function: Length of list c (see below).

Invariant P: The linked list looks as shown below (using a new variable c). In words: the first part of the list has been reversed, and c contains the name of the first node of the list that has not been reversed. Note that b or c (or both) could be null, meaning that list b or c (or both) is empty. For example, if c is null, b contains the name of the reversed list.
Question 3 (25 points). Algorithmic complexity

Part (a). Let \( f(n) = 2n^2 + 3n \).
Let \( g(n) = 5n + 10 \).
Prove that \( g(n) \) is \( O(f(n)) \).

Part (b). What is the order of execution time for the algorithm shown at the bottom of the page? Explain your reasoning -- just giving an answer is not appropriate and will not receive full credit, even if it is correct.

```c
/* Store in x the number of sections of equal elements of array b[0..n-1], for n>=0. For example, for the array b = (3,5,3,3,3,6,6,6), the value 4 is stored in x: there is a section of (one) 3’s, then a section of 5’s, then a section of 3’s, and finally a section of 6’s. */
int x= 0; int k= 0;
// invariant P: 0 <= k <= n, and
x = number of sections of equal elements in b[0..k-1], and
the following holds: k=0 or k=n or b[k-1] != b[k]
// bound function: n-k
while (k != n) {
    x= x+1;
    // Let v be the value in b[k] at this point. Increase k until either k = n or b[k] != v
    k= k+1;
    // invariant: 1 <= k <= n, and
    x = number of sections of equal elements in b[0..k-1],
    // bound function: n-k
    while (k != n && b[k-1] = b[k]) {
        k= k+1;
    }
} // R: x = number of sections of equal elements in b[0..n-1]
```
Question 4. Miscellaneous.

Part (a) Given is a doubly linked list with head and tail. p is the name of a node in the list (not the head or tail node). Write the code to remove p from the list. Remember, each node has fields prev and next; we assume you know what these are for.

Part (b) Nodes of a binary tree are of the class-type shown to the right. Write function printPreorder:

```java
// Print the elements of tree t, in preorder
class BinaryNode {
    public BinaryNode left;
    public Object element;
    public BinaryNode right;
}
```

Part (c) What is the load factor of a hash table? In looking for a value in a hash table, what is the expected number of probes if the load factor is 1/2? Why is quadratic probing preferred over linear probing?
1 (a) Precondition: n >= 0
Postcondition: b[0..n-1] is sorted

int k= 0;
// {inv: 0 <= k <= n and
// b[0..k-1] is sorted and
// b[0..k-1] <= b[k..n-1]}
// {bound function: n-k}
while (k != n) {
    Store in j the index of smallest value in b[k..n-1];
    Swap b[k] and b[j];
    k= k+1;
}

1 (b) To sort an array of size n, mergesort is always O(n log(n)). So the worst-case and best-case times are the same: O(n log(n)). Quicksort’s worst-case time is O(n^2), which happens when method partition always makes one of its two partitions empty. Its best-case time is O(n log(n)), which happens when method partition always makes the two partitions the same size.

2 (c) The binary search loop always cuts the size of the segment still being looked at it half, and it terminates only when the size is 1. Hence, it’s worst-case and best-case times are the same: O(log n).

2 (b) c= b; b= null;
    // inv: as shown on prelim 2
    // bound function: size of list c
    while (c != null) {
        ListNode t= c;
        c= c.next;
        t.next= b;
        b= t;
    }

3 (a) The definition is: g(n) is O(f(n)) if there exists a c>0 and N0 > 0 such that for all n>= N0, g(n) < c*f(n). In this case, we calculate as follows:

\[
g(n) < c*f(n) \\
= \quad \text{<substitute for g and f>}
\]
\[
5n + 10 <= 2cn^2 + 3cn
= \quad \text{<arithmetic>}
\]
\[
0 <= 2cn^2 + 3cn - 5n - 10
= \quad \text{<choose c = 1; arithmetic>}
\]
\[
0 <= 2n^2 + -2n - 10
\]

The last formula is true if n>10. So, we choose N0 = 10 and c=1.

3 (b) Variable k is increased by 1 in two places, and n is not changed. Since k <= n always holds, the maximum number of times k is increased is at most n. Therefore, in total, the inner loop body is executed at most n times. Therefore, in spite of the nested loops, the algorithm is O(n).

4 (a) p.prev.next= p.next;
p.next.prev= p.prev;

4 (b) // Print the elements of tree t, in preorder

public void printPreorder(BinaryNode t) {
    if (t == null)
        return;
    System.out.println("" + t.element);
    printPreorder(t.left);
    printPreorder(t.right);
}

4 (c) The load factor of a hash table is the ratio of the number of elements actually in it to the size of the array. The expected number of probes if the load factor is 1/2 is 2. Linear probing has primary clustering --values that hash to k (say) and k+1 and k+2 tend to cluster together, making the time it takes to find one of them longer. Quadratic probing eliminates such primary clustering.