CS1010 Tutorial 8

Slides by Ryan Tan Yu

Agenda for Today

- Problem Set 23
- Problem Set 24
- Assignment 6 Comments
- Assignment 7 Hints

Problem Set 23

Problem 23.1(a)

Write the code for performing Binary Search using loops. Identify the loop invariant and explain why the code works.

Problem 23.1(a)

What is the loop invariant?

```
long search(const long list[], long len, long q)
    long i = 0, j = len - 1;
    while (i <= j)
        long mid = (i + j)/2;
        if (list[mid] == q)
            return mid;
        else if (list[mid] > q)
            j = mid - 1;
        else
            i = mid + 1;
    return -1;
```

Problem 23.1(a)

```
long search(const long list[], long len, long q)
    long i = 0, j = len - 1;
    while (i <= j)
        long mid = (i + j)/2;
        if (list[mid] == q)
            return mid;
        else if (list[mid] > q)
            j = mid - 1;
        else
            i = mid + 1;
    return -1;
```

- `{ q is not in list[0]..list[i-1] and list[j+1]..list[n-1] }` (Binary Search)
- `{ list[0...i-1] < q < list[j+1...n-1] }` (Our invariant)

Problem 23.1(b)

Instead of returning the position of the query q, modify binary search such that it returns the insert position of q as described below:

- A position k, such that $A[k] \leq q \leq A[k+1]$
- -1 if $q < A[0]$
- $n-1$ if $q > A[n-1]$

Problem 23.1(b)

- How should we modify the code below?

```
long search(const long list[], long len, long q)
    long i = 0, j = len - 1;
    while (i <= j)
        long mid = (i + j)/2;
        if (list[mid] == q)
            return mid;
        else if (list[mid] > q)
            j = mid - 1;
        else
            i = mid + 1;
    return -1;
```

Problem 23.1(b)

- Just change `return` statement

```
long search(const long list[], long len, long q)
    long i = 0, j = len - 1;
    while (i <= j)
        long mid = (i + j)/2;
        if (list[mid] == q)
            return mid;
        else if (list[mid] > q)
            j = mid - 1;
        else
            i = mid + 1;
    return i - 1; // just change this line
```

Problem Set 24

Problem 24.1

Modify Bubble Sort to stop the sorting procedure when a pass through the array does not lead to any swapping.

```
void bubble_pass(long last, long a[])
{
    for (long i = 0; i < last; i += 1) {
        if (a[i] > a[i+1]) {
            swap(a, i, i+1);
        }
    }
}

void bubble_sort(long n, long a[]) {
    for (long last = n - 1; last > 0; last -= 1) {
        bubble_pass(last, a);
    }
}
```

Problem 24.1

```
bool bubble_pass(long last, long a[])
{
    bool swapped = false; // flag variable
    for (long i = 0; i < last; i += 1) {
        if (a[i] > a[i+1]) {
            swapped = true; // set to true only if a swap occurs
            swap(a, i, i+1);
        }
    }
    return swapped;
}

void bubble_sort(long n, long a[]) {
    for (long last = n - 1; last > 0; last -= 1) {
        if (!bubble_pass(last, a)) { // check if swap has happened
            return;
        }
    }
}
```

Problem 24.2

Suppose the input list to insertion sort is **already sorted**. What is the running time of insertion sort?

```
void insert(long a[], long curr)
    long i = curr - 1;
    long temp = a[curr];
    while (i >= 0 && temp < a[i])
        a[i+1] = a[i];
        i -= 1;
    a[i+1] = temp;

void insertion_sort(long n, long a[])
    for (long curr = 1; curr < n; curr += 1)
        insert(a, curr);
```

Problem 24.2

```
void insert(long a[], long curr)
    long i = curr - 1;
    long temp = a[curr];
    while (i >= 0 && temp < a[i]) // temp < a[i] is always false for sorted input
        a[i+1] = a[i];
        i -= 1;
    a[i+1] = temp; // this function is O(1)

void insertion_sort(long n, long a[])
    for (long curr = 1; curr < n; curr += 1)
        insert(a, curr);
```

- Since `insert` takes $O(1)$ for all indices
- The number of iterations made is the number of iterations made by the loop in `insertion_sort` - `insertion_sort` on a sorted input is $O(n)$

Problem 24.2

Suppose the input list to insertion sort is **inversely sorted**. What is the running time of insertion sort?

```
void insert(long a[], long curr)
    long i = curr - 1;
    long temp = a[curr];
    while (i >= 0 && temp < a[i])
        a[i+1] = a[i];
        i -= 1;
    a[i+1] = temp;

void insertion_sort(long n, long a[])
    for (long curr = 1; curr < n; curr += 1)
        insert(a, curr);
```

Problem 24.2

```
void insert(long a[], long curr)
    long i = curr - 1;
    long temp = a[curr];
    while (i >= 0 && temp < a[i])
        a[i+1] = a[i];
        i -= 1;
    a[i+1] = temp;

void insertion_sort(long n, long a[])
    for (long curr = 1; curr < n; curr += 1)
        insert(a, curr);
```

- On each call to `insert`, the number of iterations made depends on the value of `curr`
- `curr` is dependent on the value of `n`

Problem Set 24.2

- The number of iterations of the loop in `insert` is $[1, 2, 3, ..., n-1]$
- The running time of InsertionSort on an inversely sorted input is

$$\sum_{i=1}^{n-1} i = \frac{(n-1)(n)}{2} \in O(n^2)$$

Problem Set 24.3

What is the loop invariant for the loop in the function insert?

```
void insert(long a[], long curr)
    long i = curr - 1;
    long temp = a[curr];
    while (i >= 0 && temp < a[i])
        a[i+1] = a[i];
        i -= 1;
    a[i+1] = temp;

void insertion_sort(long n, long a[])
    for (long curr = 1; curr < n; curr += 1)
        insert(a, curr);
```

Problem Set 24.3

What is the loop invariant for the loop in the function insert?

```
void insert(long a[], long curr)
    long i = curr - 1;
    long temp = a[curr];
    // { temp <= a[j], for all i+1 <= j <= curr }
    while (i >= 0 && temp < a[i])
        a[i+1] = a[i];
        i -= 1;
        // { temp <= a[j], for all i+1 <= j <= curr }
    // { temp <= a[j], for all i+1 <= j <= curr }
    a[i+1] = temp;

void insertion_sort(long n, long a[])
    for (long curr = 1; curr < n; curr += 1)
        insert(a, curr);
```

Problem Set 24.4

- Sometimes, comparison is more expensive than assignment
  - e.g comparing two strings is more expensive than assigning a string to a variable
- Reduce the number of comparisons during InsertionSort by doing the following
- **Note that we can sort any list where an *ordering* is defined between elements**

```
repeat
    take the first element X from unsorted partition
    use binary search to find the correct position to insert X
    insert X into the right place
until the unsorted partition is empty.
```

Problem Set 24.4

```
void insert(long a[], long curr)
    long i = curr - 1;
    long temp = a[curr];
    long pos = search(a, curr - 1, temp); // include search
    while (i > pos) // no longer require comparison
        a[i+1] = a[i];
        i -= 1;
    a[i+1] = temp;

void insertion_sort(long n, long a[])
    for (long curr = 1; curr < n; curr += 1)
        insert(a, curr);
```

Assignment 6 Comments

Add

- Generally well-done, none with greater than $O(n)$
- Supposed to use a whole lot of assertions
    - Some assertions were very complex
    - It's usually sufficient to just check `0 <= i && i < length`
- Common mistakes
    - Not allocating enough memory
    - Not assigning `'\0` to the end of the string, causing reads into unallocated memory with `cs1010_println_string`
    - There's no need for very complex logic if `result[0] == '0'`, can just print `result + 1`

Frequency

- Generally well-done

Permutation

- Generally well-done given the *sliding window* hint
- Intended $O(n + k)$ algorithm:
  - Let h be the overall string, and s be the substring
  - Initialise frequency array F_s of s
  - Initialise frequency array F_h, of the first substring of length $|s|$ in h
  - Check if $F_s = F_h$
    - If so, return true
    - Otherwise, decrement h_0 from F_h and increment $h_{|s|}$ in F_h
  - Continue for every possible substring of length $|s|$ in h
  - If no substring of h is a permutation of s, return false

Permutation

Assignment 7 Hints

Question 1 - Peak

- The data always follows a "mountain" format
- When you access an element, consider the element to the left and to the right
- Examples
  - Note that $m = (i + j) \div 2$
  - $A[m-1] < A[m] < A[m+1]$ implies the peak may be on the *right*
  - $A[m-1] > A[m] \geq A[m+1]$ implies the peak may be on the *left*
- How many cases are there?
- Hint: there's definitely more than 3
- $O(n)$ may occur if the data is entirely flat

Question 2 - Sort

- The hint is to scan the input from the *front* and *back*
- **You don't need to sort the array in-place**
    - Just assign to a new array of the same size
- Hint: google the "Merge" algorithm

Question 3 - Inversion

- Hint for an $O(n)$ algorithm is "*sort*"
- The hint is referring to the previous question
- You should scan the array from the front and back
- Another way is to scan from the peak, outwards
    - Use the $O(n)$ FindMax algorithm