CS1010

Evan Tay | *evantay@comp.nus.edu.sg*

https://github.com/DigiPie/cs1010_tut_c09

Today's plan

- Unit 23: Binary Search
- Unit 24: Sorting
  - *Problem Set 24*
- Programming Exercise
- Consultation

UNIT 23
BINARY SEARCH

Recap.

The most straightforward approach – Linear Search

```
1  long search(long n, const long list[n], long q) {
2    for (long i = 0; i < n; i += 1) {
3      if (list[i] == q) {
4        return i;
5    }
6    return -1;
7  }
```

- Time complexity: O(*n*)
  - *Cannot be sure q does not exist till n elements are checked*

The most straightforward approach – Linear Search

```
1  long search(long n, const long list[n], long q) {
2    for (long i = 0; i < n; i += 1) {
3      if (list[i] == q) {
4        return i;
5    }
6    return -1;
7  }
```

- But what if the list is already sorted?

What if the list is already sorted?

- If the list is sorted in increasing order
  - *Pick a random element x from the list.*
    - Any element to the left of x <= x
    - Any element to the right of x >= x.
- If the list is sorted in decreasing order
  - *Pick a random element x from the list.*
    - Any element to the left of x >= x
    - Any element to the right of x <= x.

What if the list is already sorted?

- If the list is sorted in increasing order
  - *Pick a random element x from the list.*
    - Any element to the left of x <= x
    - Any element to the right of x >= x.
- If looking for q,
  - *If q == x, return position of x*
  - *Else if q < x, search left-side*
  - *Else (given q > x), search right-side*

Binary Search

```
7  long search(const long list[], long i, long j, long q) {
8    if (i > j) {
9      return -1;
10   }
11   long mid = (i+j)/2;
12   if (list[mid] == q) {
13     return mid;
14   } else if (list[mid] > q) {
15     return search(list, i, mid-1, q);
16   }
17   return search(list, mid+1, j, q);
18 }
```

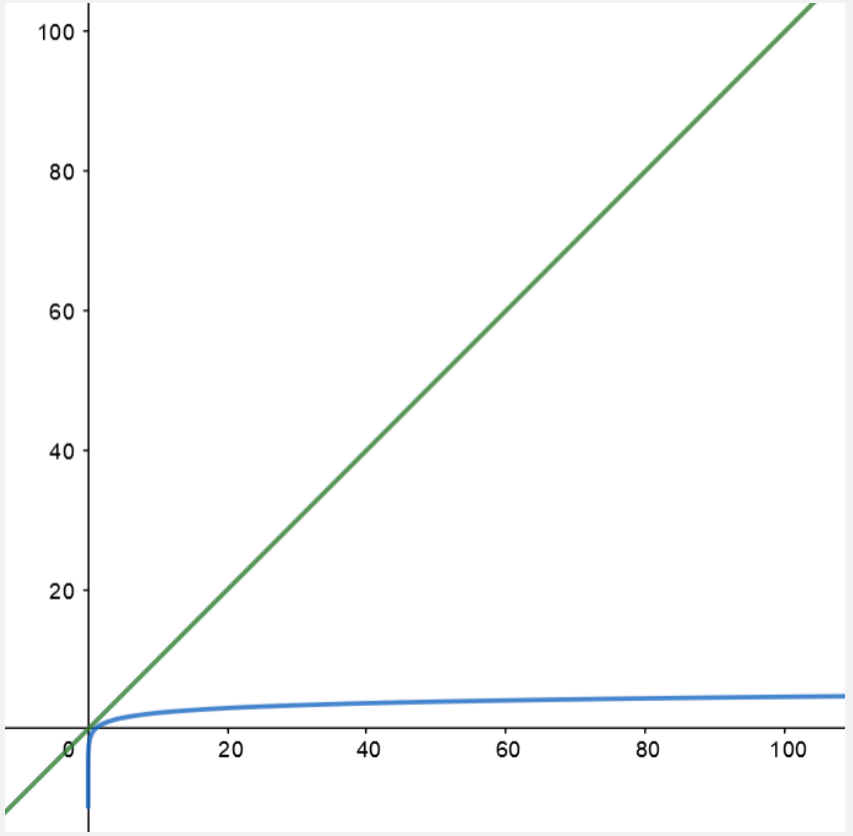
Binary Search

```
7  long search(const long list[], long i, long j, long q) {
8    if (i > j) {
9      return -1;
10   }
11   long mid = (i+j)/2;
12   if (list[mid] == q) {
13     return mid;
14   } else if (list[mid] > q) {
15     return search(list, i, mid-1, q);
16   }
17   return search(list, mid+1, j, q);
18 }
```

- Time complexity: $O(log_2 n)$

Linear Search vs Binary Search

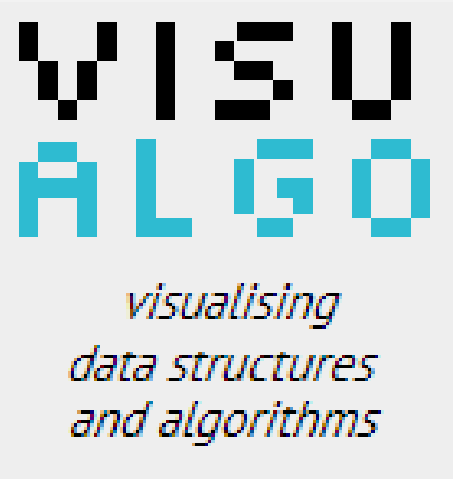
- Binary Search is much more efficient than Linear Search.
  - *But **only works if list is sorted.***



UNIT 24
SORTING

Recap. PS 24.1. PS 24.2. PS 24.3.

Visualisation of sorting



- https://visualgo.net/en/sorting

UNIT 24
SORTING

Recap. PS 24.1. PS 24.2. PS 24.3.

Problem Set 24.1

```
1   void bubble_pass(long last, long a[])
2   {
3     for (long i = 0; i < last; i += 1) {
4       if (a[i] > a[i+1]) {
5         swap(a, i, i+1);
6       }
7     }
8   }
9
10  void bubble_sort(long n, long a[n]) {
11    for (long last = n - 1; last > 0; last -= 1) {
12      bubble_pass(last, a);
13    }
14  }
```

- In this implementation, we always make $n-1$ passes. But it is possible to terminate early when a pass through the array does not lead to any swapping. Modify the code above to achieve this optimization.

Problem Set 24.1

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

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

In this implementation, we always make $n-1$ passes. But it is possible to terminate early when a pass through the array does not lead to any swapping. Modify the code above to achieve this optimization.

UNIT 24
SORTING

Recap. PS 24.1. PS 24.2. PS 24.3.

Problem Set 24.2

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

```
1  void insert(long a[], long curr)
2  {
3    long i = curr - 1;
4    long temp = a[curr];
5    while (temp < a[i] && i >= 0) {
6      a[i+1] = a[i];
7      i -= 1;
8    }
9    a[i+1] = temp;
10 }
11
12 void insertion_sort(long n, long a[n]) {
13   for (long curr = 1; curr < n; curr += 1) {
14     insert(a, curr);
15   }
16 }
```

Problem Set 24.2

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

- *O(n)*

```
1  void insert(long a[], long curr)
2  {
3    long i = curr - 1;
4    long temp = a[curr];
5    while (temp < a[i] && i >= 0) {
6      a[i+1] = a[i];
7      i -= 1;
8    }
9    a[i+1] = temp;
10 }
11
12 void insertion_sort(long n, long a[n]) {
13   for (long curr = 1; curr < n; curr += 1) {
14     insert(a, curr);
15   }
16 }
```

Problem Set 24.2

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

```
1  void insert(long a[], long curr)
2  {
3    long i = curr - 1;
4    long temp = a[curr];
5    while (temp < a[i] && i >= 0) {
6      a[i+1] = a[i];
7      i -= 1;
8    }
9    a[i+1] = temp;
10 }
11
12 void insertion_sort(long n, long a[n]) {
13   for (long curr = 1; curr < n; curr += 1) {
14     insert(a, curr);
15   }
16 }
```

Problem Set 24.2

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

- *O(n^2)*

```
1  void insert(long a[], long curr)
2  {
3    long i = curr - 1;
4    long temp = a[curr];
5    while (temp < a[i] && i >= 0) {
6      a[i+1] = a[i];
7      i -= 1;
8    }
9    a[i+1] = temp;
10 }
11
12 void insertion_sort(long n, long a[n]) {
13   for (long curr = 1; curr < n; curr += 1) {
14     insert(a, curr);
15   }
16 }
```

Problem Set 24.2

Insertion Sort

- Best time complexity:
  - *O(n)*
- Worst time complexity:
  - *O(n^2)*

```
1  void insert(long a[], long curr)
2  {
3    long i = curr - 1;
4    long temp = a[curr];
5    while (temp < a[i] && i >= 0) {
6      a[i+1] = a[i];
7      i -= 1;
8    }
9    a[i+1] = temp;
10 }
11
12 void insertion_sort(long n, long a[n]) {
13   for (long curr = 1; curr < n; curr += 1) {
14     insert(a, curr);
15   }
16 }
```

UNIT 24
SORTING

Recap. PS 24.1. PS 24.2. PS 24.3.

Problem Set 24.3

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

```
1  void insert(long a[], long curr)
2  {
3    long i = curr - 1;
4    long temp = a[curr];
5    while (temp < a[i] && i >= 0) {
6      a[i+1] = a[i];
7      i -= 1;
8    }
9    a[i+1] = temp;
10 }
11
12 void insertion_sort(long n, long a[n]) {
13   for (long curr = 1; curr < n; curr += 1) {
14     insert(a, curr);
15   }
16 }
```

Problem Set 24.3

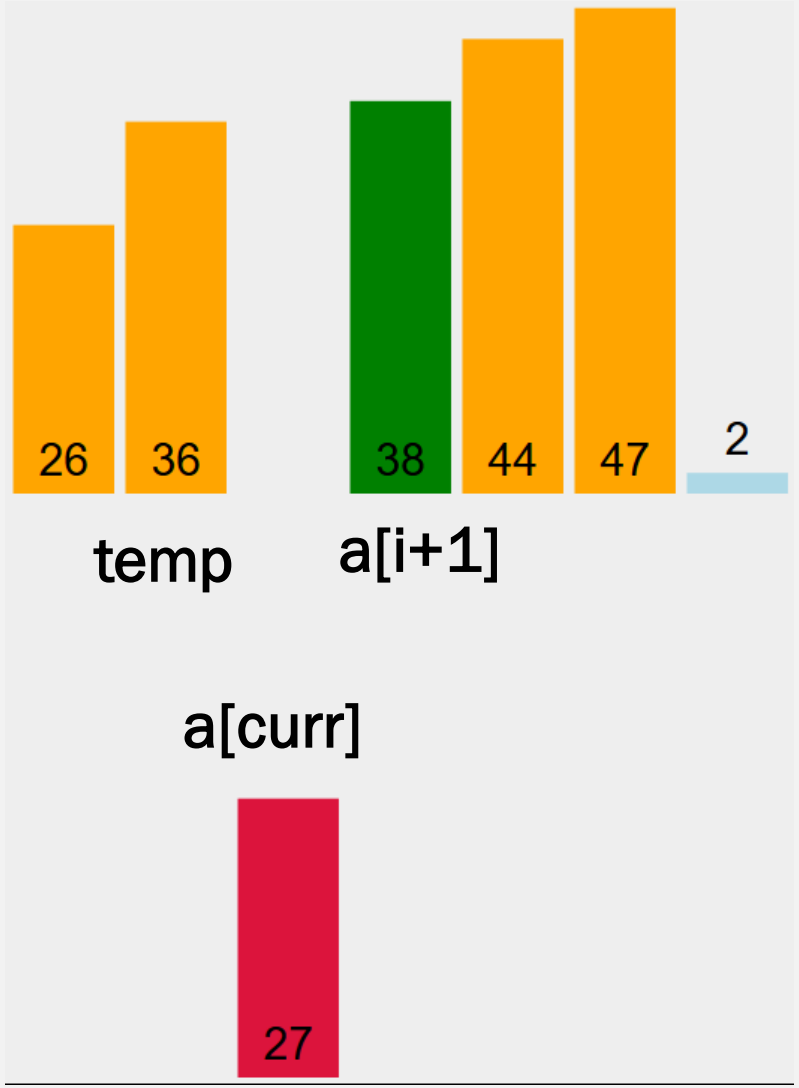
- What is the loop invariant for the loop in the function insert?
- temp <= a[i+1]..a[curr].

```
1  void insert(long a[], long curr)
2  {
3    long i = curr - 1;
4    long temp = a[curr];
5    while (temp < a[i] && i >= 0) {
6      a[i+1] = a[i];
7      i -= 1;
8    }
9    a[i+1] = temp;
10 }
11
12 void insertion_sort(long n, long a[n]) {
13   for (long curr = 1; curr < n; curr += 1) {
14     insert(a, curr);
15   }
16 }
```

evantay@comp.nus.edu.sg

Problem Set 24.3

- What is the loop invariant for the loop in the function insert?
- temp <= a[i+1]..a[curr].



IN-CLASS EXERCISES

1. Implement Binary Search with Loop
2. Modify binary search so that it returns a position *k* such that a[k] <= q <= a[k+1]
   - *-1 if q < a[0]*
   - *n-1 if q > a[n-1].*
     - Basically, this is the position that we should insert k in to keep the array sorted.
3. Combine insertion sort with binary search

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

THE END

https://github.com/DigiPie/cs1010_tut_c09