CS 211: Linear Search, Binary Search, Sorting

Chris Kauffman

Week 12

Searching Linearly

- Return the first index of `key` in array `arr`
- Return -1 if not present
- No assumptions on ordering of `arr`
- What assumptions are needed about available methods?

For Ints

```
public static int linearSearch(int arr[], int key)
```

For Strings

```
public static int linearSearch(String arr[], String key)
```

For Foos

```
public static int linearSearch(Foo arr[], Foo key)
```

Linear Search for Primitives and Objects

```
public static
int linearSearch(int arr[],
                 int key)
{
  for(int i=0; i<arr.length; i++){
    if(arr[i] == key){
      return i;
    }
  }
  return -1;
}
```

```
public static
int linearSearch(Object arr[],
                 Object key)
{
  for(int i=0; i<arr.length; i++){
    if(arr[i].equals( key )){
      return i;
    }
  }
  return -1;
}
```

Questions

- Could you go faster with a sorted array?
- If so how?
- How much work is it to sort an array?

Binary Search

- A faster way to search sorted arrays
- Array must be sorted: linear search doesn't require that
- Like searching for a word in a dictionary: repeated halving of search space (look left or look right)
- Let's write it:

```
// Iterative
// Use binary search to locate an a given integer in an array
public static int binarySearch(int a[], int key)

// Easy-to-use recursive version which calls a helper
public static int binarySearchR(int a[], int key){
  return binarySearchR(a,key,0,a.length-1);
}

// Recursive
// Helper method which does work of searching, repeatedly
// halving search area by adjusting left/right
public static int binarySearchR(int a[], int key,
                                int left, int right)
```

Binary Search: Iterative

```
// Iterative: Use binary search to
// locate integer key in array a
public static
int binarySearch(int a[], int key){
  int left=0, right=a.length-1;
  int mid = 0;
  while(left <= right){
    mid = (left+right)/2;
    if(key == a[mid]){
      return mid;
    }else if(key < a[mid]){
      right = mid-1;
    }
    else{
      left = mid+1;
    }
  }
  return -1;
}
```

Demonstration

```
key: 6  -> index 2
key: 13 -> not found
key: 2  -> index 0
key: 25 -> not found

array a:
| 0 | 1 | 2 | 3 |  4 |  5 |  6 |  7 |  8 |  9 |
|---+---+---+---+----+----+----+----+----+----|
| 2 | 3 | 6 | 9 | 11 | 12 | 14 | 15 | 17 | 21 |
```

Binary Search: Recursive

```
// Easy-to-use recursive version which calls a helper
public static int binarySearchR(int a[], int key){
  return binarySearchR(a,key,0,a.length-1);
}
// Helper method which does work of searching
public static int binarySearchR(int a[], int key,
                                int left, int right){
  if(left > right){
    return -1;
  }
  int mid = (left+right)/2;
  if(key == a[mid]){
    return mid;
  }else if(key < a[mid]){
    return
      binarySearchR(a,key,left,mid-1);
  }
  else{
    return
      binarySearchR(a,key,mid+1,right);
  }
}
```

Speed of Linear vs Binary Search

Specific Size

- Input array 1024 elements
- `key` is not present
- How many steps to determine `key` is not there?
  - For linear search?
  - For binary search?

General Size

- Input array N elements
- `key` is not present
- How many steps to determine `key` is not there?
  - For linear search?
  - For binary search?

Quick Review: Search

- Describe one way to search for data in an array; include any assumptions necessary for this process to work
- Describe a fundamentally different way
- Compare these two approaches according to their *worst case runtime complexity*

Sorting

For binary search to work, must have sorted input

- How do I get `ints` or `Strings` or anything else sorted?
- What do you know so far about sorting?
- What you should know as Computer Scientist is ...

How does sorting Actually Work?

- Tons of CPU time devoted to sorting
- Records in databases
- Rows/Columns in spreadsheets
- We will discuss Selection Sort, simple and inefficient
  - (BJP 13.3)
- May have time to discuss Merge Sort which is more efficient
  - (BJP 13.4)
- Get hungry for better sorting algorithms

Other courses

- CS 211: Start basic discussions
- CS 310: Framework for analysis (Big-O notation)
- CS 483: Analyze detailed algorithms

Selection Sort

- Dead simple sorting
- Repeatedly look for the minimum element in right part of array
- Swap min element with last element of left part of array

```
public static void selectionSort(int[] a) {
  for (int i = 0; i < a.length - 1; i++) {
    int smallest = i;
    for (int j = i + 1; j < a.length; j++) {
      if (a[j] < a[smallest]) {
        smallest = j;
      }
    }
    swap(a, i, smallest);
  }
}
public static void swap(int[] a, int i, int j) {
  int temp = a[i]; a[i] = a[j]; a[j] = temp;
}
```

An A/V Intensive Demonstration

- Selection Sort by Timo Bingmann, hosted on YouTube
- Also worth a watch: Gypsy Dance Demo of Selection Sort on YouTube

Demonstration of Selection Sort

```
a = [ 19 18 21 12 14 16 13 17 42 31]
       0  1  2  3  4  5  6  7  8  9
Outer i = 0, smallest = 0 (19)
Inner
j = 1, smallest = 1 (18)
j = 2, smallest = 1 (18)
j = 3, smallest = 3 (12)
j = 4, smallest = 3 (12)
...
j = 9, smallest = 3 (12)
swap(a, i,smallest)
a = [ 12 18 21 19 14 16 13 17 42 31]
       0  1  2  3  4  5  6  7  8  9
Outer i = 1, smallest = 1 (18)
Inner
j = 2, smallest = 1 (18)
j = 3, smallest = 1 (18)
j = 4, smallest = 4 (14)
j = 5, smallest = 4 (14)
...
j = 9, smallest = 6 (13)
swap(a, i,smallest)
a = [ 12 13 21 19 14 16 18 17 42 31]
       0  1  2  3  4  5  6  7  8  9
Outer i = 2, smallest = 2 (21) ...
```

```
public static
void selectionSort(int[] a) {
  for(int i=0; i<a.length-1; i++){
    int smallest = i;
    for(int j=i+1; j<a.length; j++){
      if (a[j] < a[smallest]) {
        smallest = j;
      }
    }
    swap(a, i, smallest);
  }
}
```

Show the next few outer iterations

Runtime of Selection sort

```
public static void selectionSort(int[] a) {
  for (int i = 0; i < a.length - 1; i++) {
    int smallest = i;
    for (int j = i + 1; j < a.length; j++) {
      if (a[j] < a[smallest]) {
        smallest = j;
      }
    }
    swap(a, i, smallest);
  }
}
public static void swap(int[] a, int i, int j) {
  int temp = a[i]; a[i] = a[j]; a[j] = temp;
}
```

- Each inner loop iteration does a constant amount of work: Fetch array elements, compare numbers, set variables
- Good approximation of runtime: total inner loop iterations
- Total inner iterations for array size 10? size 50? size N?

Speed and Alternatives

Selection Sort is

- Easy to code, easy to understand
- Has terrible runtime performance
- $O(N^2)$ WORST case performance
- $O(N^2)$ BEST case performance
  - Pre-sorted array still takes $O(N^2)$ operations

Alternatives

- Many sorting algorithms exist
- Visualization and Comparison of Sorting Algorithms by Viktor Bohush, hosted on YouTube
- Good sorting algorithms have $O(N \log N)$ worst case performance
- What algorithm does `Arrays.sort()` and `Collections.sort()` use?

Asymptotic Speeds To Remember

Input Data

Array of size N

Search

- Linear search: $O(N)$
- Binary search: $O(\log N)$
- Hash tables: $O(1)$

Sort

- Selection sort: $O(N^2)$
- Insertion sort: $O(N^2)$
- Merge sort: $O(N \log N)$
- Quick sort: $O(N \log N)$
- Radix sort: $O(N)$ via cheating

Generalizing Binary Search

What would change in the binary search code if `String` were the operative type rather than `int`?

```
public static
int binarySearch(String a[],
                 String key)
```

```
public static
int binarySearch(int a[],
                 int key)
{
  int left=0, right=a.length-1;
  int mid = 0;
  while(left <= right){
    mid = (left+right)/2;
    if(key == a[mid]){
      return mid;
    }else if(key < a[mid]){
      right = mid-1;
    }
    else{
      left = mid+1;
    }
  }
  return -1;
}
```

Generalized Searching and Sorting

Search/Sort in the Java tied to two interfaces

`interface Comparable<T>` : Objects compare to each other

- Class has function `int compareTo(T y)`

  ```
  if(x.compareTo(y) < 0){...
  ```

- `x.compareTo(y)`: Returns "x minus y"
  - Negative for `x` before `y`
  - 0 for equal
  - Positive for `x` after `y`
- Note: Not always -1/0/+1: why not?

`interface Comparator<T>` : Judge of two other objects

- Class has `compare(T x,T y)`: Returns "x minus y"
- Neg/0/Pos numbers for ordering

  ```
  Comparator<String> cmp = ...;
  if( cmp.compare(x,y) < 0){...
  ```

Quick Review: Sorting

Sorting

- Describe how one sorting algorithm works
- Give its worst-case runtime complexity
- What is the runtime complexity of the best sorting algorithms?
- Given an example of one sorting algorithm that achieves this complexity
- What sorting algorithm does `Collections.sort(..)` use anyway?

Comparing

- What's one way a java class can be made to work with functions like `Arrays.binarySearch(..)` and `Collections.sort(..)`?
- Are there any other ways?

Generalized Binary Search

Adapt String version of binary search to

```
public static <T extends Comparable<T>>
int binarySearch(T a[], T key)
{
  ...
}
```

Works with any `Comparable` thing: `String`, `Integer`, `Person`

```
public static
int binarySearch(int a[],
                 int key)
{
  int left=0, right=a.length-1;
  int mid = 0;
  while(left <= right){
    mid = (left+right)/2;
    if(key == a[mid]){
      return mid;
    }else if(key < a[mid]){
      right = mid-1;
    }
    else{
      left = mid+1;
    }
  }
  return -1;
}
```

Visit you Local Library

Arrays and Collections have sort() methods for Comparable stuff and which take a Comparator

From Arrays

```
// Uses compareTo of comparable
static void sort(Object[] a)

// Uses comparator
static <T> void sort(T[] a, Comparator<? super T> c)
```

From Collections

```
// Uses compareTo of comparable
static <T extends Comparable<? super T>> void sort(List<T> list)

// Uses comparator
static <T> void sort(List<T> list, Comparator<? super T> c)
```

Comparable Exercise

Person

Write a class `Person` which implements the `Comparable` interface

```
class Person implements Comparable<Person>{
  public Person(String first, String last);
  public String toString();
  public int compareTo(Person other);
}
```

- Must have last method to satisfy `Comparable` interface
- Sort by last name then first name
- String already has a `compareTo` so this is easy

Try this with Generalized Binary Search

Comparators and Sorting

- Sorting algs in `Arrays` and `Collections` are generalized
- Comparator provides a powerful way to sort in new ways
  - Just define `int compare(T x, T y)` and sort
- Example: Reverse numeric comparison:

```
import java.util.*;
class RevComp implements Comparator<Integer>{
  public int compare(Integer x, Integer y){
    return y-x;
  }
}
public class RevComparator{
  public static void main(String args[]){
    Integer a[] = {4, 5, 9, 2, 3, 1, 8};
    Comparator<Integer> cmp = new RevComp();
    Arrays.sort(a, cmp);
    System.out.println(Arrays.toString(a));
    // [9, 8, 5, 4, 3, 2, 1]
  }
}
```

Exercise: Odds First

Define a `Comparator` on Integers which sorts odds before evens with odd numbers in order within the first section and even numbers in order within the second section.

```
public static void main(String args[]){
  Comparator<Integer> cmp = new OddsThenEvens();
  Integer a[];

  a = new Integer[]{4, 5, 9, 2, 3, 1, 8};
  System.out.println(Arrays.toString(a));
  Arrays.sort(a,cmp);
  System.out.printf("%s\n\n",Arrays.toString(a));
  // [1, 3, 5, 9, 2, 4, 8]

  a = new Integer[]{2048, 1024, 5096, 128, 9999};
  System.out.println(Arrays.toString(a));
  Arrays.sort(a,cmp);
  System.out.printf("%s\n\n",Arrays.toString(a));
  // [9999, 128, 1024, 2048, 5096]
}
```

Optional: Merge Sort

- Fast sorting algorithm: $O(N \log N)$
- Exploits recursion
- Principles are simple but implementation is non-trivial
- Variants have different properties: out-of place vs in-place
- Good culmination of CS 211 last two weeks of material

Exercise: Merge

```
public static void merge(int[] result, int[] a, int[] b)
```

- a and b are sorted arrays, may not be same size
- Copy elements from a and b into result so that result is sorted
- Assume: `result.length = =a.length + b.length`
- Target Runtime Complexity: $O(N)$ where N is `result.length`

Example

```
int [] a = {1, 6, 7, 9}
int [] b = {0, 2, 3, 4, 8}
int [] result = new int[a.length+b.length];

merge(result, a, b);

print(result)
// [0, 1, 2, 3, 4, 6, 7, 8, 9]
```

Merge Answers

Textbook

```
public static
void merge(int[] result,
           int[] a, int[] b) {
  int i1 = 0;
  int i2 = 0;
  for(int i = 0; i < result.length; i++){
    if(i2 >= b.length ||
       (i1 < a.length && a[i1] <= b[i2])){
      result[i] = a[i1];
      i1++;
    }
    else {
      result[i] = b[i2];
      i2++;
    }
  }
}
```

CK preferred

```
public static
void merge(int[] result,
           int[] a, int[] b) {
  int ai=0, bi=0;
  for(int ri=0; ri<result.length; ri++){
    if(ai >= a.length){
      result[ri] = b[bi];
      bi++;
    }
    else if(bi >= b.length){
      result[ri] = a[ai];
      ai++;
    }
    else if(a[ai]<=b[bi]){
      result[ri] = a[ai];
      ai++;
    }
    else{
      result[ri] = b[bi];
      bi++;
    }
  }
}
```

Recursive Application

```
public static void mergeSort(int[] a) {
  if (a.length <= 1) {
    return;
  }
  int[] left  = Arrays.copyOfRange(a, 0, a.length/2);
  int[] right = Arrays.copyOfRange(a, (a.length/2), a.length);

  mergeSort(left);
  mergeSort(right);

  merge(a, left, right);
}
```

- An array if 1 element is sorted
- If bigger, chop array into two halves
- Recursively sort left and right halves
- Merge the sorted results into the original array

Demonstrate

```
public static void mergeSort(int[] a) {
  if (a.length <= 1) {
    return;
  }
  int[] left  = Arrays.copyOfRange(a, 0, a.length/2);
  int[] right = Arrays.copyOfRange(a, (a.length/2), a.length);

  mergeSort(left);
  mergeSort(right);

  merge(a, left, right);
}
```

Show Execution for

```
int [] a = {14, 32, 67, 76, 23, 41, 58, 85};
mergeSort(list);
```

`mergeSortDebug()` in `CKMergeSort.java` is useful to see this

Computational Complexity

- What is the complexity of `mergeSort()`
- Could you prove it?
- What's one huge disadvantage of this version of merge sort?

In-Place Merge Sort

- Making copies of arrays takes time and memory
- Current version is an out-of-place merge sort
- For and array of size 1,000,000, need `left` and `right` arrays which total another 1,000,000 units of memory - BAD!
- Prefer an in-place version to save memory
- Cost: Implementation complexity
- Examine: In-place merge sort ported from C++ STL