Stack and Recursion

Based on the notes from David Fernandez-Baca and Steve Kautz

Bryn Mawr College
CS206 Intro to Data Structures

The Java Virtual Machine

- Java is designed to be **platform-independent**. To achieve this, every class in a Java program is translated by the compiler into a separate **class file** (extension .class), written in an intermediate language that is understood by the **Java Virtual Machine** (JVM).
- The instructions of this language are called **bytecodes**, because each of them typically occupies one byte.
- JVM is an abstract machine that is (typically) implemented in software, and is then executed by a specific machine.
- The JVM is **stack-based**.

Program Execution

At runtime, the compiled class files are loaded independently into the JVM and then execution begins. Java stores stuff in two separate pools of memory:

- The **heap** stores all objects, including all arrays, and all class variables (i.e., those declared "static").
- The **stack** stores all local variables, including all parameters.

When a method is called, the Java Virtual Machine creates a **stack frame** (also known as an **activation record**) for the method and pushes it onto the stack. The stack frame stores the parameters and local variables for that method.

Program Execution (cont.)

At any given moment, the state of a method's execution consists of:

- a reference to the instruction stream for the method, which we can think of as a byte array, along with an index for the next instruction to be executed, called the **instruction pointer** or IP,
- an **operand stack**, along with an index representing the top of the stack, called the stack pointer or SP, and
- an area for **local variables**. The bytecodes refer to local variables by index, starting at index 0. For a static method with n parameters, local variables 0 through n – 1 are always the parameters, in order of their appearance in the method signature.

Executing Recursive Program 1: Binary Search Revisited

Recall the setting for binary search: We are given an int array arr sorted from least to greatest —for instance,

arr = (-3, -2, 0, 0, 1, 5, 5).

We want to search the array for a given value v. If we find v, we return its array index; otherwise, we return -1.

Base Cases

Binary search checks the middle array element first. If v is lesser, we recursively search in the left half of the array. If v is greater, we recursively search in the right half. The recursion has two base cases.

- If v equals the middle element, return its index; in the example above, we return 4.
- If we try to search a subarray of length zero, the array does not contain v, and we return -1.

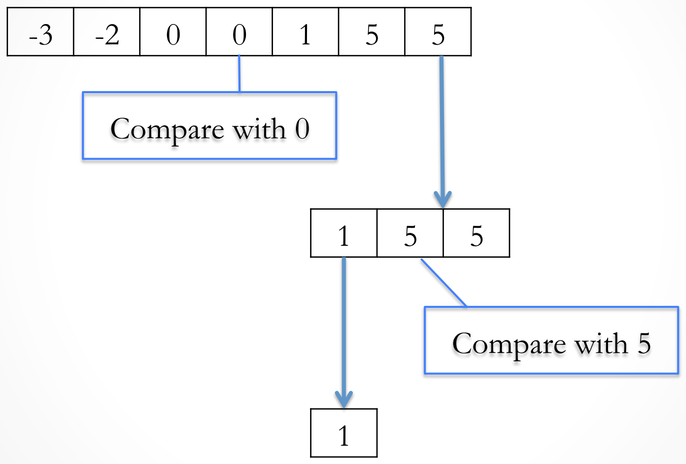
Since we are cutting the possibilities in half at each step, the time complexity is O(log n).

Implementation

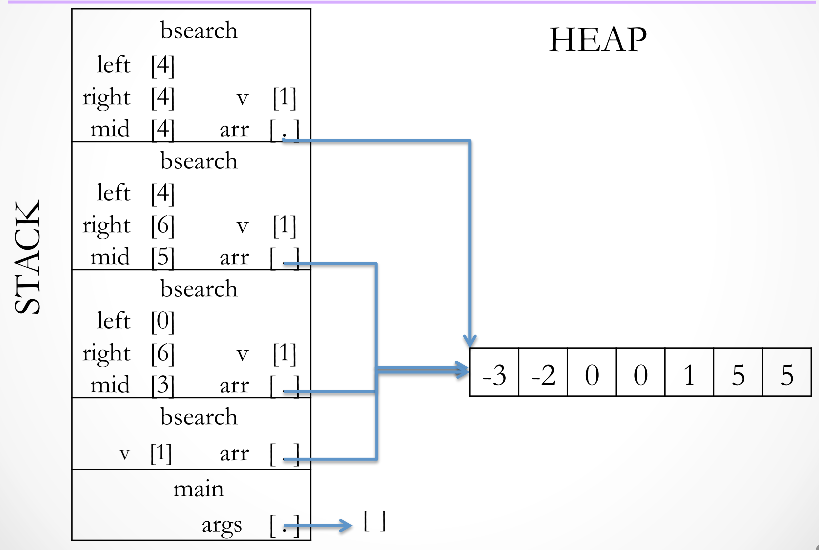
```
private static int bsearch (int[] arr, int left, int right, int v) {
    if (left > right) { return -1; }
    int mid = (left + right) / 2;
    if (v == arr[mid]) { return mid; }
    else if (v < arr[mid]) { return bsearch(arr, left, mid-1, v); }
    else { return bsearch(arr, mid+1, right, v); }
}

public static int bsearch(int[] arr, int v) {
    return bsearch(arr, 0, arr.length-1, v);
}
```

Execution of a Search for v=1

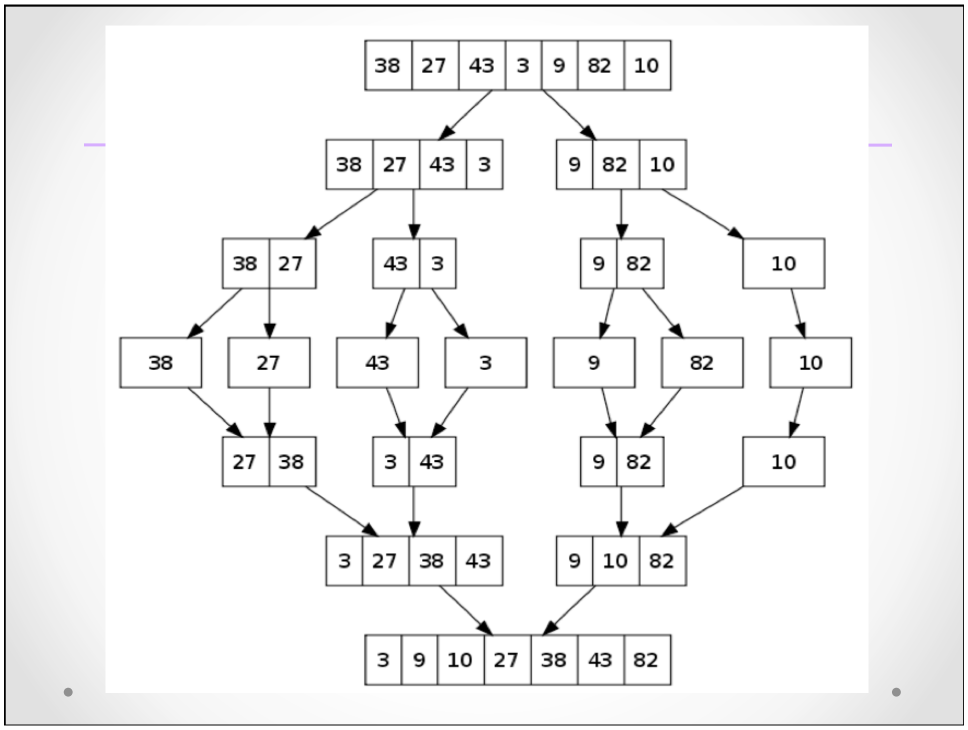


Stack and Heap



Executing Recursive Program 2: Merge Sort

```
MergeSort(A, p, r):
        if(p< r)
                q=(p+r)/2
                MergeSort(A,p,q)
                MergeSort(A,q+1,r)
                Merge(A,p,q,r)
```


38 27 43 3 9 82 10
38 27 43 3
9 82 10
38 27
43 3
9 82
10
38
27
43
3
9
82
10
27 38
3 43
9 82
10
3 27 38 43
9 10 82
3 9 10 27 38 43 82