Execution of a Recursive Algorithm

To execute a program, it first needs to be translated to machine code that the computer can understand. This translation is performed by the compilers.

Once translated to machine code, each instruction of the program is stored in memory. Each memory cell has a unique address, so each instruction of the program is stored in a unique memory address.

Consider the recursive version of the binary search algorithm:

```
Algorithm   BinarySearch (L,x, first, last )
Input:    Array L of size n and value x
Output: Position i, 0 ≤ i < n, such that L[i] = x, if x
            in L, or -1, if  x not in L
  if  first  > last  then return  -1
  else  {
     mid  ←  (first +last )/2

     if  x = L[mid ] then return  mid
     else if  x < L[mid ] then
               return  BinarySearch (L,x,first,mid -1)   (A1)
        else return BinarySearch (L,x,mid +1,last )  (A2)
```

We have marked the two recursive calls with (A1) and (A2). This means that when translated to machine code, the code for performing the first recursive call is stored in some memory address that we will denote as A1 and the code for performing the second recursive call is stored in a memory address that we will denote as A2.

Consider the following algorithm that invokes BinarySearch:

Algorithm *main* ()

L ← array storing values [4, 7, 11]

pos ← BinarySearch (*L*,11,0,2) (A3)

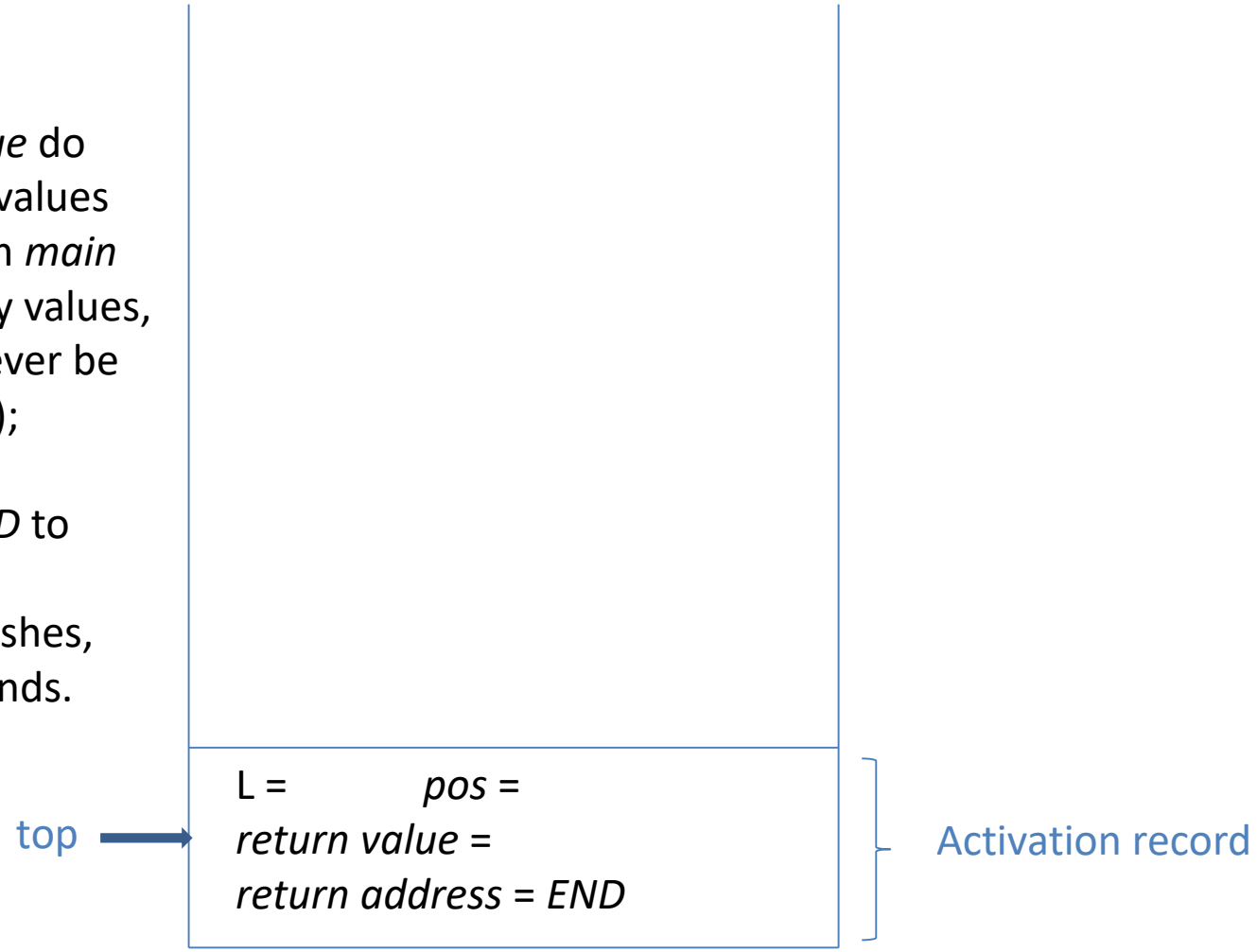
Print *pos*

We have marked the invocation to BinarySearch with (A3) to indicate that the machine code that implements the invocation is stored in memory in an address that will be denoted A3.

When executing a program on a computer, part of the memory of the computer is used as an *execution stack* or *runtime stack*. The execution stack is needed to be able to execute algorithms that invoke other algorithms. When algorithm main is invoked, part of the execution stack is reserved to store the local variables, parameters, return address and return value. This portion of the execution stack is called an *activation record* or a *frame*. Since algorithm main uses 2 local variables, *L* and *pos*, the activation record for algorithm main will look like this (we assume that array *L* is stored in memory in address (A4)):

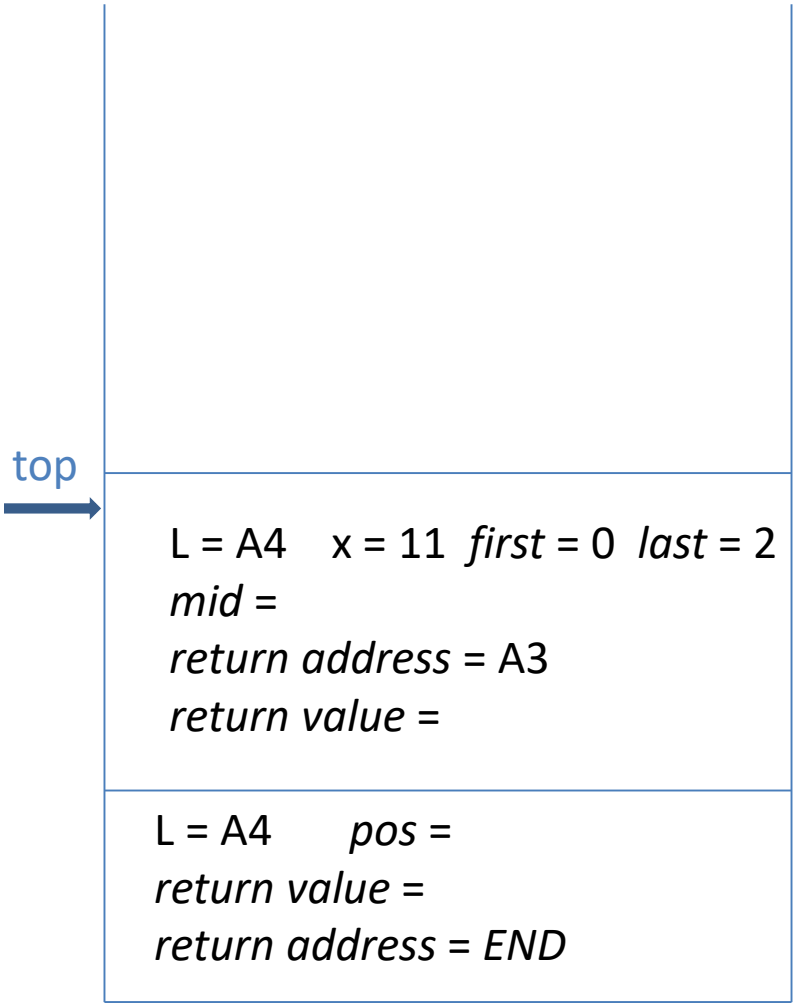
Execution stack

Note that in the activation record *L*, *pos* and *return value* do have assigned any values yet (since algorithm *main* does not return any values, *return value* will never be assigned any value); *return address* has been set to *END* to indicate that when algorithm *main* finishes, the program also ends.



Execution stack

Once the activation record has been created the execution of algorithm *main* starts. First, an array storing 3 values: 4, 7, and 11 is created and the address is stored in L. Let the address of the array be A4; this value is stored in the activation
record.
Next, algorithm BinarySearch is invoked. Every time
that an algorithm is invoked a new activation record is created. Note that the activation record for BinarySearch needs to store the parameters *(L*, *x*, *first*, *last)* the local variables (*mid)*, the return
address and the return value.
Once the activation record in created, the values of
the parameters are stored and the return address.
At this point the activation record will look like the figure on the right.

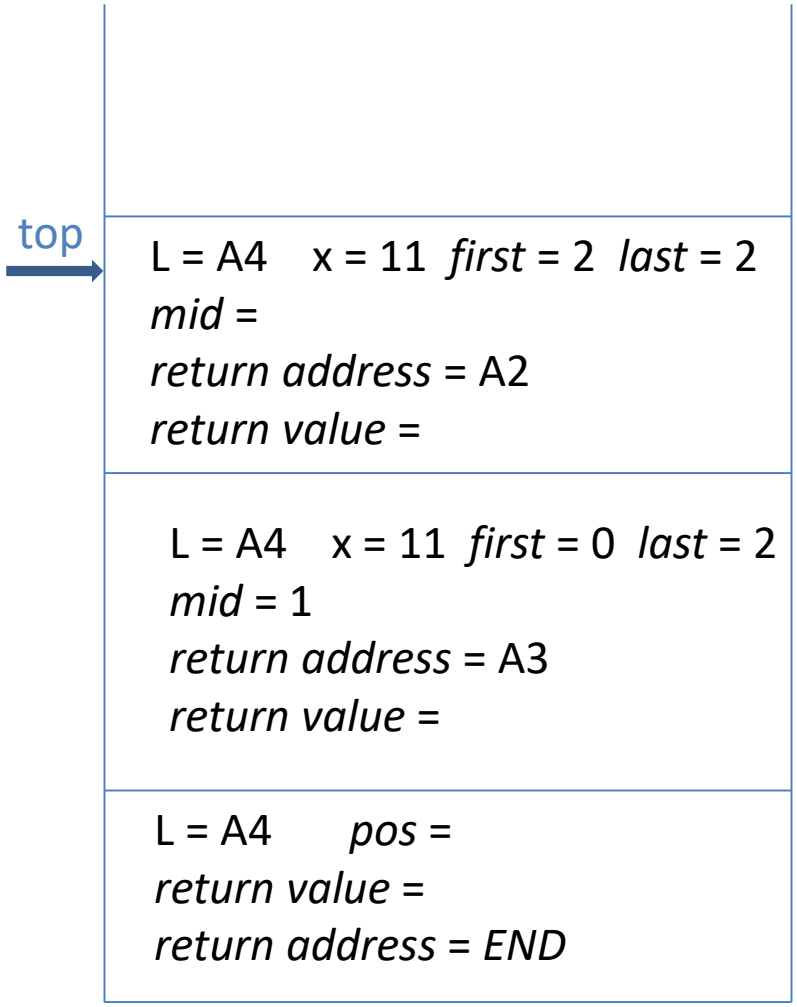


Execution stack

Execution stack

Once the activation record has been created the execution of algorithm BinarySearch starts: Since condition (*first* > *last*) is false the algorithm computes mid (= (0+2)/2 = 1). Then as x is different from L[1] and furthermore x is larger than L[1] then the second recursive call is invoked.
A new activation record is created and the values of the parameters are stored in it as shown in the figure.

Note that the only activation record that is active at any moment during the execution of the program is the one at the top of the execution stack. This record is marked by *top*.



Execution stack

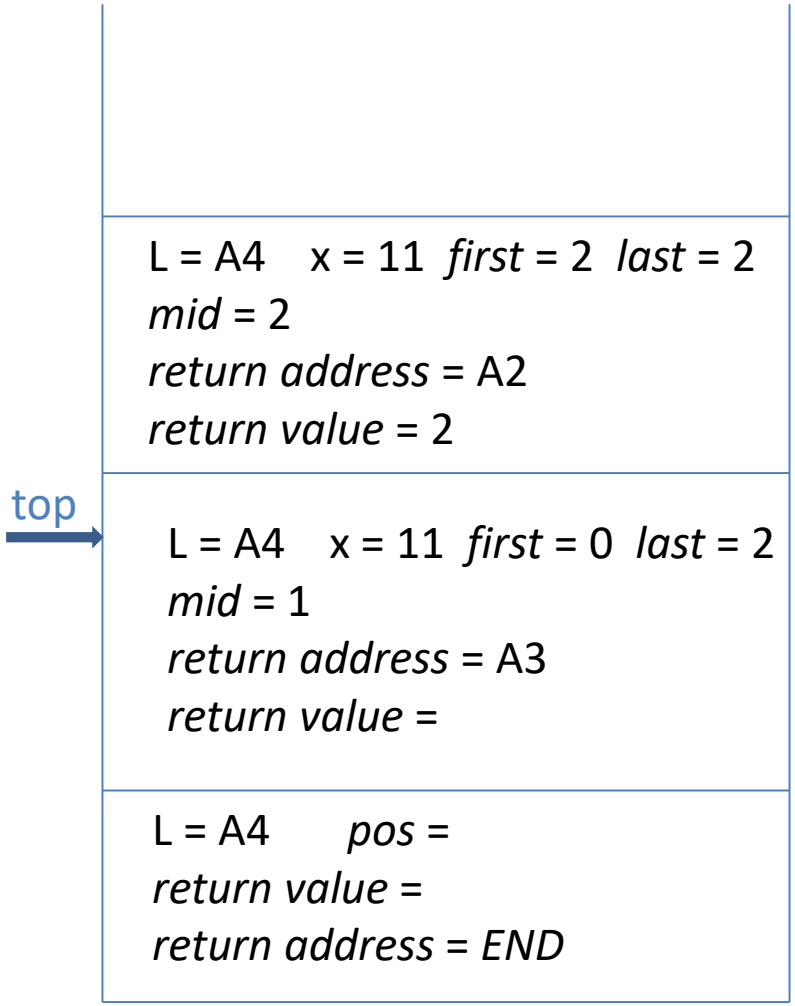
Execution stack

Once the activation record has been created the execution of algorithm BinarySearch starts: Since *first* = *last* then the first else statement is executed and the value of mid is computed (mid = (2+2)/2 = 2). Then x is compared to L[2] and since both values are the same the algorithm executes the statement

return *mid*

When the return statement is executed, first the value of *mid* is stored in the activation record under *return value* as shown in the figure. Then the activation record is popped from the execution stack by moving *top* to the second activation record as shown.

To determine where the execution of the program should continue after a return statement, the activation record on top of the current one is examined and the return address is extracted. In this case the return address is A2.



Execution stack

Execution stack

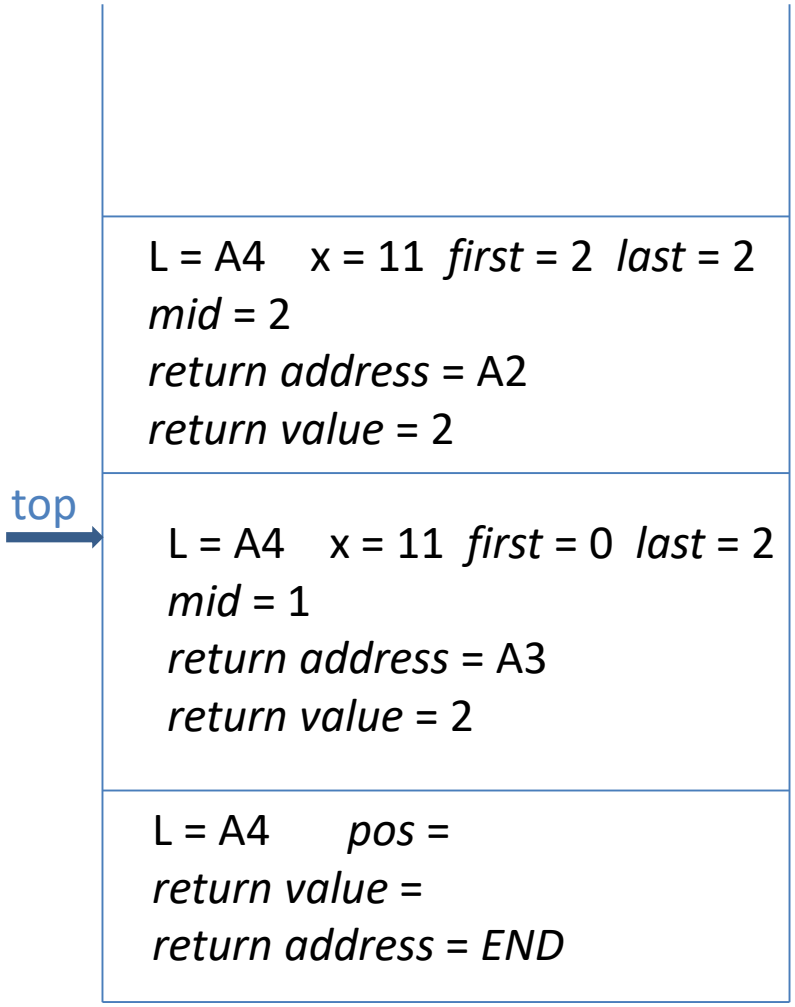
Since A2 is the address for the statement

else return BinarySearch (L,x,mid +1,last) (A2)

then execution continues from here. This recursive call has just completed and the value that it returned (2) is stored in the activation record that was just popped from the execution stack. Hence, the above statement is equivalent to

else return 2 (A2)

To execute this statement, the value 2 is first stored in the active activation record (as shown in the figure).



Execution stack

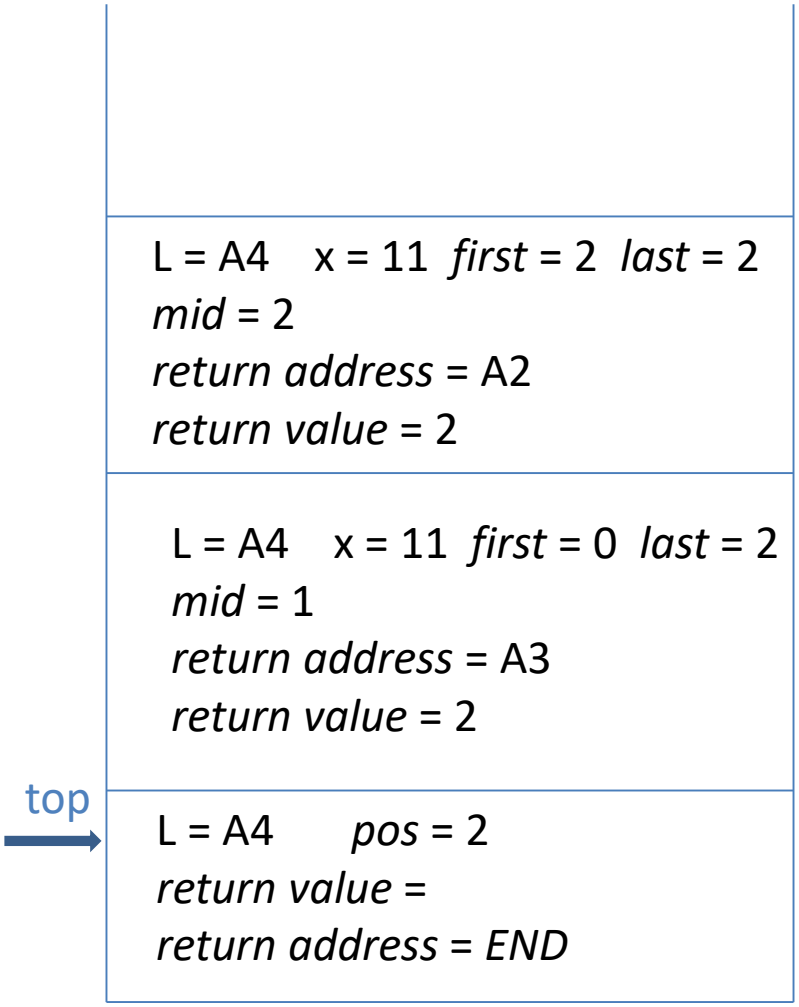
Execution stack

Then the **return** statement ends the execution of algorithm BinarySearch, so the current activation record is popped from the execution stack as shown in the figure.

The return address in the activation record that was just popped is A3, which means that the execution of the program continues at algorithm *main*:

pos ← BinarySearch (*L*,11,0,2) (A3)

The value returned by BinarySearch is 2 (this value is stored in the activation record just popped), and this value is then stored in *pos* as show in the figure.
The value of *pos* is printed and then the algorithm terminates. When algorithm *main* ends the current activation record is popped from the stack. Since the return address in the current activation record is *END*, then the entire program ends.



Execution stack