* Discuss at least **two** of the following topics on Recursion for your initial post. Provide a code example to elaborate on your thoughts.
  + Sort using recursion.
  + Binary search using recursion.
  + Discuss how recursive method invokes are handled in a call stack.
  + How to print a directory structure file list using recursion.

As we learned after reading our latest chapter and reviewing our resources, recursion is a method that invokes itself directly or indirectly. Something to always remember when using recursive algorithms is that they must include a base case. Otherwise, an infinite recursion will occur. Recursion allows us to break a problem into smaller problems. It also allows us to solve problems that would be more difficult, if not impossible, to solve without it. Some may be confused or not understand the differences between recursion and iteration, but it boils down to that recursion is repetition without a loop (Liang, 2019/2025). Recursion allows us to perform all sorts of abilities, such as sorting, binary search, invoking a call stack, and printing a directory structure file list. However, I will focus on sorting and binary search using recursion for this discussion.

The selection sort locates the smallest element within a list, swapping it with the first (Liang, 2019/2025). This same process occurs again until only one element in the list is left. This is also known as a merge sort and helps to “divide and conquer code” since it divides an array, sorts the first part, then the second part, merges both, and returns the sorted array (GeeksforGeeks, 2013).

Here is an example of sorting using recursion: (GeeksforGeeks, 2013)

// Java program for Merge Sort

class MergeSort {

// Merges two subarrays of a[]

void merge(int a[], int l, int m, int r)

{

int n1 = m - l + 1;

int n2 = r - m;

int L[] = new int[n1];

int R[] = new int[n2];

for (int i = 0; i < n1; ++i)

L[i] = a[l + i];

for (int j = 0; j < n2; ++j)

R[j] = a[m + 1 + j];

// Merge the temp arrays

// Initial indexes of first and second subarrays

int i = 0, j = 0;

int k = l;

while (i < n1 && j < n2) {

if (L[i] <= R[j]) {

a[k] = L[i];

i++;

}

else {

a[k] = R[j];

j++;

}

k++;

}

while (i < n1) {

a[k] = L[i];

i++;

k++;

}

while (j < n2) {

a[k] = R[j];

j++;

k++;

}

}

// Main function that sorts a[l..r] using

// merge()

void sort(int a[], int l, int r)

{

if (l < r) {

int m = (l + r) / 2;

// Sort first and second halves

sort(a, l, m);

sort(a, m + 1, r);

// Merge the sorted halves

merge(a, l, m, r);

}

}

// Driver method

public static void main(String args[])

{

int a[] = { 12, 11, 13, 5, 6, 7 };

// Calling of Merge Sort

MergeSort ob = new MergeSort();

ob.sort(a, 0, a.length - 1);

int n = a.length;

for (int i = 0; i < n; ++i)

System.out.print(a[i] + " ");

}

}

A binary search starts by comparing the key with the element in the middle of the array (Liang, 2019/2025). This requires the array to be created in increasing order (Liang, 2019/2025). A binary search goes off of a low and high for the array.

Here is an example of a Binary Search: (GeeksforGeeks, 2014)

// Java Program to Illustrate

// Iterative Binary Search

// Main class

// BinarySearch

class GFG {

// Method 1

// Returns index of x

// if it is present in arr[],

// else return -1

int binarySearch(int arr[], int x)

{

int l = 0, r = arr.length - 1;

// Checking element in whole array

while (l <= r) {

int m = l + (r - l) / 2;

// Check if x is present at mid

if (arr[m] == x)

return m;

// If x greater, ignore left half

if (arr[m] < x)

l = m + 1;

// If x is smaller,

// element is on left side

// so ignore right half

else

r = m - 1;

}

// If we reach here,

// element is not present

return -1;

}

// Method 2

// Main driver method

public static void main(String args[])

{

GFG ob = new GFG();

// Input array

int arr[] = { 2, 3, 4, 10, 40 };

// Length of array

int n = arr.length;

// Element to be checked if present or not

int x = 10;

// Calling the method 1 and

// storing result

int result = ob.binarySearch(arr, x);

// Element present

if (result == -1)

// Print statement

System.out.println("Element not present");

// Element not present

else

// Print statement

System.out.println("Element found at index "

+ result);

}

}

**References**

GeeksforGeeks. (2013, March 15). *Java Program for Merge Sort*. GeeksforGeeks. https://www.geeksforgeeks.org/java-program-for-merge-sort/

GeeksforGeeks. (2014, January 28). *Java Program for Binary Search (Recursive and Iterative)*. GeeksforGeeks. https://www.geeksforgeeks.org/java-program-for-binary-search-recursive-and-iterative/

Liang, D. Y. (2025). *Introduction to Java Programming and Data Structures: comprehensive version*. Pearson. (Original work published 2019)

**Assignment Requirements and Grading:**

* + 1. An initial post of approximately 250 words is due by **Thursday, 11:59 p.m., CST**.
    2. For the initial post to be considered substantive, it should be at least 250 words in length and fully cover the topics being presented. Single-sentence definitions or responses will not be awarded points.
    3. Submit your post by clicking on the assignment link above, then Create Thread. You must create a thread in order to view your peers' posts. Tip: Create your post in a Word document and then copy and paste your work into the thread.
    4. A minimum of three (3) responses, to the original threads of other students, of 100-200 words each are due by **Sunday, 11:59 p.m., CST**.
    5. To view the rubric grading criteria, click on the following link: [Discussion Board Grading Rubric.](https://content.bellevue.edu/cst/csd/rubricdbv3.pdf)

Lea, you did a great job on your discussion board for this week! You are correct that there are different ways to sort using recursion. In my research, I also found a quick sort and heap sort option in Java. Quick sort utilizes a pivot element, which breaks the array into two, then sorts into the subarrays created. The next recursive sorting option is heap sort. Heap sort requires the heapify() function to be utilized to classify it as a recursive action. Binary search and merge sort really do show the power of recursion when it comes to simplifying problems.

Hey, Samir. I think you did an excellent job of explaining how binary search works using recursion and how recursive method invokes are handled in a call stack. The code you provided for each topic accurately depicts how to apply it when programming in Java. Something important to remember when using binary search is that the elements in the sorted array must be in increasing order. A low and high value are used in the search. I am glad you mentioned how the call stack uses a last in, first out format because it is vital to remember when coding.

Hi, Nardos! I really enjoyed reading your post for this week. You accurately explained how the recursive method invokes are handled in a call stack and binary search. The included codes for each also perfectly showcase how recursion can handle both topics. GeeksforGeeks always has very helpful information, making it easier to understand more complex subjects like this. If you are still struggling to understand how the call stacks work in Java, I recommend re-watching the animation 18.2 in our textbook. That really helped me visualize how it works. I am sure there are also helpful videos online that you could utilize.