**Demo Exam 2023 Algorithms and Data structure (Usman A):**

**NB ! Q1 vil mest sannsynlig komme på eksamen (lurt å kunne dette!!)**

**Q1:** **Write a step-by-step algorithm that finds the largest number in a list (an array) of n numbers.**

public void largestNumberInArray(){  
 int [] arr = {1,2,3,4,5,32,7,8,9,11};  
 int n = arr.length;  
 int max = arr[0];  
 for (int i = 0; i <n ; i++) {  
 if (arr[i]>max){  
 max=arr[i];  
  
 }  
 }  
 System.*out*.println("Largest number in array:"+max);  
}

**Step by Step algorithm:**

1. Initialize a variable in the type of an int and call it: “max” and set the max variable equal to arr[0] as default value.
2. Create a loop (for loop) that goes through each element in the array starting from the first element.
3. Create an if statement inside the loop, if element is greater than the max value, set the max value equal to the element.
4. Print the max value using System.out.println statement outside the loop and the answer should be displayed 😊

**Q2:**

<https://www.interviewkickstart.com/learn/difference-between-recursion-and-iteration#:~:text=Iteration%20is%20faster%20and%20more,using%20recursion%20and%20iteration%20both>.

<https://codeahoy.com/learn/recursionjava/ch6/>

**Iterations vs Recursion:**

**Iterations:**

* Code structures that use loops is considered as iteration (or iterative codes)
* If the code repeatedly executes the set of instructions until the instructions or the loops becomes false
* Iterations has a polynomial time complexity

**Recursion:**

* In recursion the functions calls itself in an infinite loop to solve the problem
* In recursive codes or methods , we try using smaller inputs to make the problem smaller
* It uses a base condition to stop the recursion , otherwise it could lead to a running out of stack memory
* Examples : Binary Search, Merge Sort, Quick Sort (Divide and Conquer), In-order/Pre-order/Post-order Tree Traversals, DFS of Graph, Fibonacci Series, and Factorial Problems, etc

**In Common:**

* The time complexity in Both recursion and iteration is O(N)
* Even though the time complexity is the same, iteration is faster since recursion takes a lot more time when the functions are calling themselves due to overhead of functions

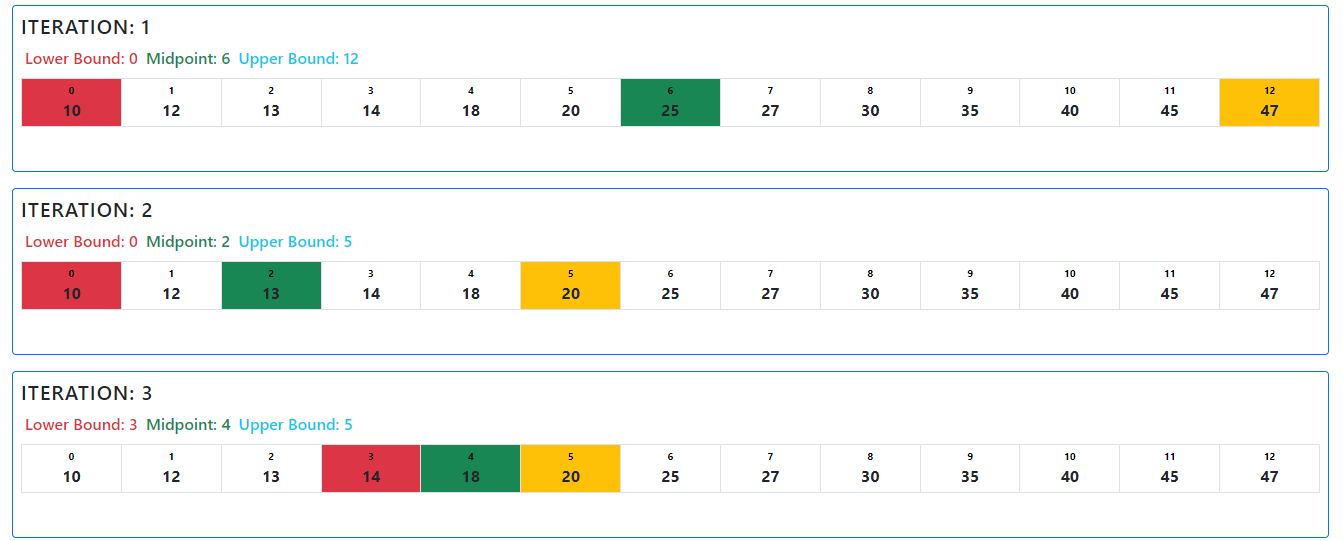
**Which is more efficient (Iteration vs Recursion)?**

As mentioned before, in recursive code the function call itself which can cause memory loss due to overhead of functions. When a function calls other functions in an infinite loop, it will lead to take more space in stack memory. In this case the stack will run out of storage over time. Iteration on the other hand doesn’t use functions when solving the problem, instead it uses loops which repeats an action. Iterative codes /method doesn’t require stack memory to work like the recursive ones, since the loop structure occurs within the method. That is why it doesn’t use any memory or lead to any overhead in the memory. This is the reason why Iteration is more efficient than recursion.

**Binary search:**

**Glemt! (Skriv en step by step algoritme til binary search)**

**Q3: i) Write an algorithm and solution to Binary Search with a Recursive version (10 points) Suppose x = 18, and we have the following array: [10,12,13,14,18,20,25,27,30,35,40,45,47]**



18 is located on index x=4

**ii) When not to use Divide and Conquer (5 points) Short answer 3-4 sentences**

We cannot use divide and conquer in cases where the array is **unsorted**. To solve problems where divide and conquer is required, we have to sort the array otherwise it can cause errors when solving the problem. Other problems that can occur is when the **subproblems** cannot be solved separately, the subproblems have to be independent from each other to get solved otherwise the divide and conquer algorithm will fail. The last example is when the size of the array or the list **is small**, in this case it’s unnecessary to use the divide and conquer algorithm since in this type of lists the solution is usually simple, so we can’t divide the array into smaller subproblems. That’s why we can’t perform divide and conquer and it becomes useless.

**Divide and Conquer is a powerful algorithmic technique that involves breaking a problem down into subproblems that are easier to solve. However, there are some situations where Divide and Conquer may not be the most appropriate approach to use. Here are a few examples:**

**1. When the subproblems cannot be solved independently: In Divide and Conquer, the subproblems are independent of each other and can be solved separately. If the subproblems are interdependent and cannot be solved separately, then Divide and Conquer may not be the best approach.**

**2. When the problem size is small: Divide and Conquer is most effective when the problem can be divided into smaller subproblems. If the problem size is already small, then the overhead of dividing and combining the subproblems may outweigh the benefits.**

**3. When the problem has a simple solution: If the problem has a simple and straightforward solution that does not require the use of a complex algorithm like Divide and Conquer, then it may be more efficient to use the simple solution.**

**4. When the problem requires sequential processing: Divide and Conquer involves solving subproblems in parallel, which can be beneficial for some problems. However, for problems that require sequential processing, such as string matching, Divide and Conquer may not be the best approach.**

**5. When the problem has a non-uniform input distribution: Divide and Conquer works best when the input is uniformly distributed among the subproblems. If the input is non-uniformly distributed, then some subproblems may require significantly more computation than others, resulting in inefficient use of resources.**

**Q4:**

**(i) Why has Bubble sort proved inefficient compared to other sorting algorithms? (4 points)**

Bubble sort is the simplest sorting algorithm compared to Merge sort and Quick sort. Bubble sort can’t deal with larger datasets like the Merge sort or the Quicksort can and it’s only suitable for smaller list where the efficiency isn’t important. Bubble sort is a sorting algorithm that work by repeatedly swapping elements in the unsorted list and follows the O(N^2) time complexity where as Merge Sort and Quick Sort follows the O(nlogn) time complexity. In bubble sort the list has to be unsorted to be solved, but if the list is already sorted, the bubble sort is unnecessary and cannot be performed. On the other hand, as mentioned before the Merge and the Quick sort can deal with larger lists / datasets and work by dividing the list into smaller subproblems before merging them back into a sorted list. This is why Merge sort and Quick sort has proved more efficiency than Bubble Sort.

**(ii) Sort the following list with Bubble Sort, Merge Sort and Quick Sort 123 34 189 56 150 12 9 240 (Skal man bruke java kode?? Send mail til foreleseren)**

**Bubble Sort:**

class BubbleSort {  
  
 void bubbleSort(int [] arr){  
 int n = arr.length;  
 for (int i = 0; i <n ; i++) {  
 for (int j = 0; j < n-i-1; j++) {  
 if (arr[j]>arr[j+1]){  
 int temp = arr[j];  
 arr[j] = arr[j+1];  
 arr[j+1]=temp;  
 }  
 }  
 }  
 }  
 void printArray(int [] arr){  
 int n = arr.length;  
 for (int i = 0; i <n ; i++) {  
 System.*out*.println(arr[i]);  
 }  
 }  
  
 public static void main(String[] args) {  
 int [] arr = {123, 34, 189, 56, 150 ,12 ,9 ,240};  
 BubbleSort ob = new BubbleSort();  
 System.*out*.println("Unsorted:");  
 ob.printArray(arr);  
 ob.bubbleSort(arr);  
 System.*out*.println("Sorted:");  
 ob.printArray(arr);  
 }  
}

**Quick Sort:**

package Algoritmer.exercises;  
  
import java.io.\*;  
 class QuickSort {  
  
 static void swap(int [] arr , int i , int j){  
 int temp = arr[i];  
 arr[i]=arr[j];  
 arr[j]=temp;  
 }  
   
 static int partition(int [] arr , int low , int high){  
 int pivot = arr[high];  
 int i = (low-1);  
  
 for (int j = low; j <=high-1 ; j++) {  
 if (arr[j]<pivot){  
 i++;  
 *swap*(arr , i, j);  
 }  
 }  
 *swap*(arr , i+1 ,high);  
 return (i+1);  
 }  
   
 static void quickSort(int [] arr , int low , int high){  
 if (low <high){  
 int pi =*partition*(arr , low ,high);  
   
 *quickSort*(arr , low ,pi-1);  
 *quickSort*(arr , pi+1 ,high);  
   
   
   
 }  
 }  
 static void printArray(int [] arr, int size){  
 for (int i = 0; i <size ; i++) {  
 System.*out*.println(arr[i]);  
 }  
 }  
  
 public static void main(String[] args) {  
 int [] arr = {123, 34, 189, 56, 150 ,12 ,9 ,240};  
 int n = arr.length;  
   
 System.*out*.println("Unsorted list:");  
 *printArray*(arr,n);  
 *quickSort*(arr, 0 , n-1);  
 System.*out*.println("Sorted list:");  
 *printArray*(arr , n);  
   
 }  
  
  
}

**Merge Sort:**

package Algoritmer.exercises;  
  
/\* Java program for Merge Sort \*/  
public class MergeSort {  
 void merge (int arr [] , 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]=arr[i+1];  
 }  
 for (int j = 0; j <n2 ; j++) {  
 R[j] = arr[m+1+j];  
 }  
  
 int i = 0 , j =0;  
  
 int k =l;  
  
 while (i <n1 && j < n2){  
  
 if (L[i] <=R[j]) {  
 arr[k] = L [i];  
  
 i++;  
 } else {  
 arr[k] = R [j];  
 j++;  
 }  
 k++;  
 }  
 while (i<n1){  
 arr[k] = L[i];  
 i++;  
 k++;  
 }  
  
 while (j <n2){  
 arr[k] = R[j];  
 j++;  
 k++;  
 }  
 }  
  
 void sort( int arr [] , int l , int r){  
  
 if ( l < r){  
 int m = l + (r-l)/2;  
  
 sort(arr , l , m);  
 sort(arr , m+1 , r);  
  
 merge(arr , l , m , r);  
 }  
 }  
  
 static void printArray( int [] arr){  
 int n = arr.length;  
 for (int i = 0; i <n ; i++) {  
 System.*out*.println(arr [i] + " ");  
  
 System.*out*.println();  
 }  
 }  
  
 public static void main(String[] args) {  
  
 int [] arr = {123, 34, 189, 56, 150 ,12 ,9 ,240};  
 int n = arr.length;  
 System.*out*.println("Unsorted list:");  
 *printArray*(arr);  
 System.*out*.println("Sorted list:");  
 MergeSort ms = new MergeSort();  
 ms.sort(arr , 0 ,n-1 );  
 *printArray*(arr);  
 }  
}

**(iii) provide a graphical representation of solving each of the sorting algorithms.**

(Se SkriveBoka for graphical representation av de ulike sorting algoritmene)

**Merge Sort Table format:**

**Arr = [123 34 189 56 150 12 9 240]**

|  |  |  |  |
| --- | --- | --- | --- |
| **k** | **U** | **V** | **Sorted (Result)** |
| **1** | 123,34,189,56 | 150, 12, 9, 240 | **9** |
| **2** | 123,34,189,56 | 150, 12, 9, 240 | **9,12** |
| **3** | 123,34,189,56 | 150, 12, 9, 240 | **9,12,34** |
| **4** | 123,34,189,56 | 150, 12, 9, 240 | **9,12,34,56** |
| **5** | 123,34,189,56 | 150, 12, 9, 240 | **9,12,34,56,123** |
| **6** | 123,34,189,56 | 150, 12, 9, 240 | **9,12,34,56,123,150** |
| **7** | 123,34,189,56 | 150, 12, 9, 240 | **9,12,34,56,123,150,189** |
| **----------------** | 123,34,189,56 | 150, 12, 9, 240 | **9,12,34,56,123,150,189,240**  **(Final values)** |

**Q6 : Short answers (3-4 sentences) are required for these 10 sub-questions**