Name: FOMUBAD BORISTA FONDI

Matricule: 20V2001

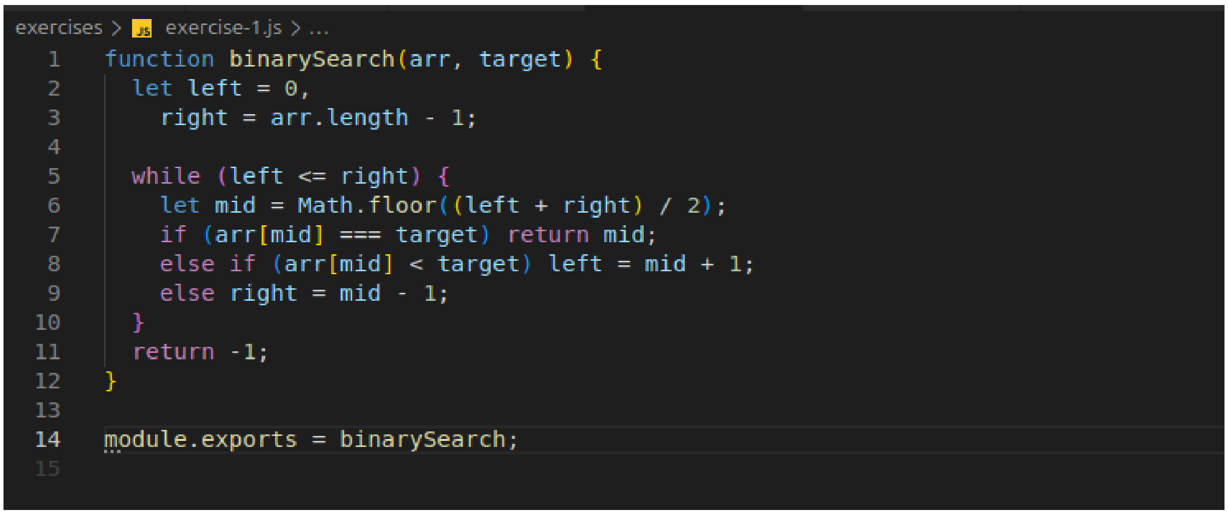
Algorithms for Problem Solving

This report presents solutions to several fundamental algorithmic problems. Each problem is solved using an efficient approach, and its performance is analyzed.

Exercise 1: Binary Search

Problem: Given a sorted list of integers, find the target value using Binary Search.

Solution: Implemented using a divide-and-conquer strategy.



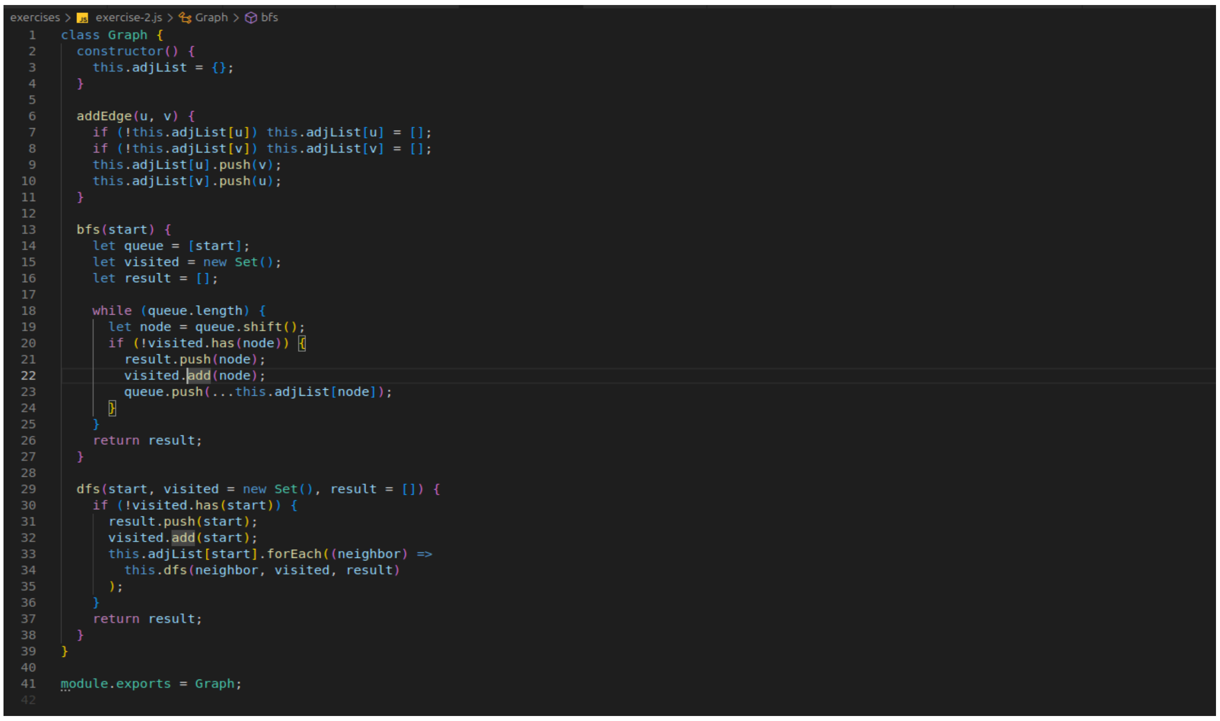
exercises > exercise-1.js > ...

```javascript
function binarySearch(arr, target) {
  let left = 0,
    right = arr.length - 1;

  while (left <= right) {
    let mid = Math.floor((left + right) / 2);
    if (arr[mid] === target) return mid;
    else if (arr[mid] < target) left = mid + 1;
    else right = mid - 1;
  }
  return -1;
}

module.exports = binarySearch;
```

Exercise 2: Graph Traversal (BFS & DFS)

Problem: Implement BFS & DFS for an undirected graph.

Solution: BFS uses a queue (level-order), DFS uses recursion (deep search).



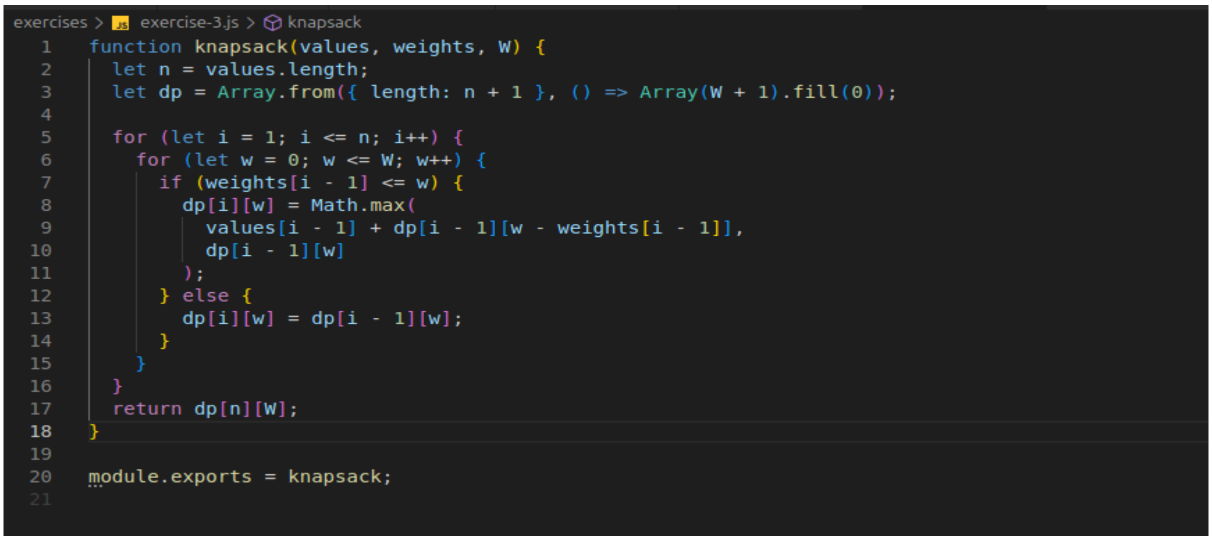
exercises > exercise-2.js > Graph > bfs

```javascript
class Graph {
  constructor() {
    this.adjList = {};
  }

  addEdge(u, v) {
    if (!this.adjList[u]) this.adjList[u] = [];
    if (!this.adjList[v]) this.adjList[v] = [];
    this.adjList[u].push(v);
    this.adjList[v].push(u);
  }

  bfs(start) {
    let queue = [start];
    let visited = new Set();
    let result = [];

    while (queue.length) {
      let node = queue.shift();
      if (!visited.has(node)) {
        result.push(node);
        visited.add(node);
        queue.push(...this.adjList[node]);
      }
    }
    return result;
  }

  dfs(start, visited = new Set(), result = []) {
    if (!visited.has(start)) {
      result.push(start);
      visited.add(start);
      this.adjList[start].forEach((neighbor) =>
        this.dfs(neighbor, visited, result)
      );
    }
    return result;
  }
}

module.exports = Graph;
```

Exercise 3: Knapsack Problem

Problem: Given items with values and weights, maximize value within weight limit W.

Solution: Dynamic Programming with a 2D DP table.



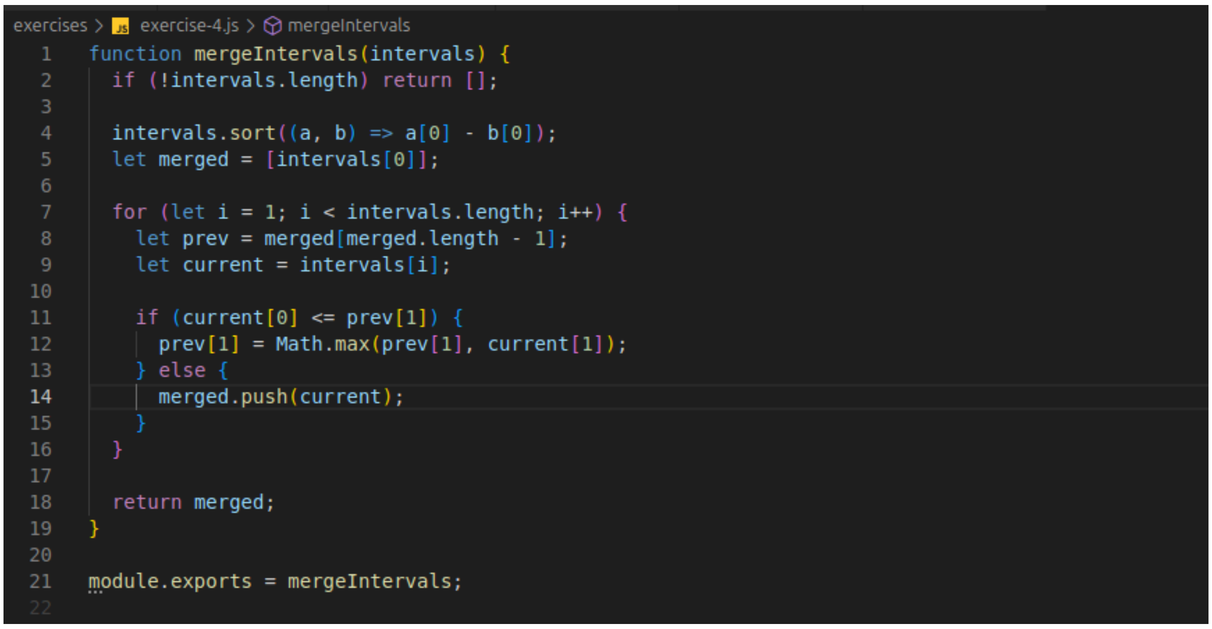
exercises > exercise-3.js > knapsack

```javascript
function knapsack(values, weights, W) {
  let n = values.length;
  let dp = Array.from({ length: n + 1 }, () => Array(W + 1).fill(0));

  for (let i = 1; i <= n; i++) {
    for (let w = 0; w <= W; w++) {
      if (weights[i - 1] <= w) {
        dp[i][w] = Math.max(
          values[i - 1] + dp[i - 1][w - weights[i - 1]],
          dp[i - 1][w]
        );
      } else {
        dp[i][w] = dp[i - 1][w];
      }
    }
  }
  return dp[n][W];
}

module.exports = knapsack;
```

Exercise 4: Merge Intervals

Problem: Given a list of intervals, merge overlapping ones.

Solution: Sort and iterate through the list.



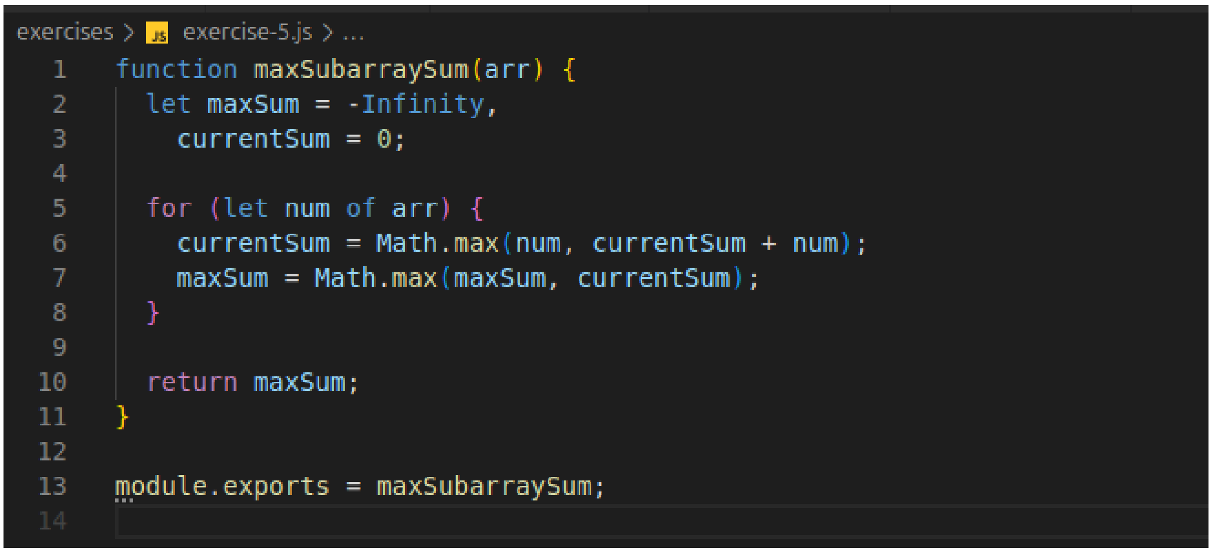
exercises > exercise-4.js > mergeIntervals

```javascript
function mergeIntervals(intervals) {
  if (!intervals.length) return [];

  intervals.sort((a, b) => a[0] - b[0]);
  let merged = [intervals[0]];

  for (let i = 1; i < intervals.length; i++) {
    let prev = merged[merged.length - 1];
    let current = intervals[i];

    if (current[0] <= prev[1]) {
      prev[1] = Math.max(prev[1], current[1]);
    } else {
      merged.push(current);
    }
  }

  return merged;
}

module.exports = mergeIntervals;
```

Exercise 5: Kadane’s Algorithm

Problem: Find the maximum subarray sum.

Solution: Dynamic Programming with an optimized approach.


exercises > exercise-5.js > ...

```javascript
function maxSubarraySum(arr) {
  let maxSum = -Infinity,
    currentSum = 0;

  for (let num of arr) {
    currentSum = Math.max(num, currentSum + num);
    maxSum = Math.max(maxSum, currentSum);
  }

  return maxSum;
}

module.exports = maxSubarraySum;
```

GitHub Repository

Link to Code Repository:
FOMUBAD-BORISTA-FONDI-SBSE