SPORTS CENTER

Midterm Exam CSAL

DESCRIPTION OF THE PROBLEM

The Sports Center project tackles the issue of efficiently navigating and purchasing from a large online inventory of sports products, aiming to simplify the search and ordering process based on price preferences. Through an interactive website equipped with advanced filtering and sorting technologies, it seeks to eliminate time-consuming searches, enhancing the shopping experience for sports enthusiasts.

DIVIDE AND CONQUER

MERGE SORT ALGORITHM

1. INITIALIZE PRICES FOR EACH PRODUCT.
2. WHEN A SORTING OPTION IS SELECTED, SORT THE PRODUCTS BASED ON PRICE.
3. IMPLEMENT MERGE SORT TO DIVIDE THE ARRAY INTO SMALLER ARRAYS, SORT THOSE ARRAYS, AND MERGE THEM BACK TOGETHER.

```
function mergeSort(arr, asc) {
    if (arr.length <= 1) return arr;

    const mid = Math.floor(arr.length / 2);
    const left = arr.slice(0, mid);
    const right = arr.slice(mid);

    mergeSort(left, asc);
    mergeSort(right, asc);

    merge(arr, left, right, asc);
}
```

CODE IMPLEMENTATION

```javascript
(function() {
    let field = document.querySelector('.items');
    let li = Array.from(field.children);
    let originalOrder = Array.from(li); // Store the original order

    function FilterProduct() {
        for (let i of li) {
            const name = i.querySelector('strong');
            const x = name.textContent;
            i.setAttribute("data-category", x);
        }

        let indicator = document.querySelector('.indicator').children;

        this.run = function() {
            for (let i = 0; i < indicator.length; i++) {
                indicator[i].onclick = function() {
                    for (let x = 0; x < indicator.length; x++) {
                        indicator[x].classList.remove('active');
                    }
                    this.classList.add('active');
                    const displayItems = this.getAttribute('data-filter');

                    for (let z = 0; z < li.length; z++) {
                        li[z].style.transform = "scale(0)";
                        setTimeout(() => {
                            li[z].style.display = "none";
                        }, 500);

                        if ((li[z].getAttribute('data-category') == displayItems) || displayItems == "all") {
                            li[z].style.transform = "scale(1)";
                            setTimeout(() => {
```

CODE IMPLEMENTATION

```
                    setTimeout(() => {
                        li[z].style.display = "none";
                    }, 500);

                    if ((li[z].getAttribute('data-category') == displayItems) || displayItems == "all") {
                        li[z].style.transform = "scale(1)";
                        setTimeout(() => {
                            li[z].style.display = "block";
                        }, 500);
                    }
                }
            };
        }
    }
}

function SortProduct() {
    let select = document.getElementById('select');
    let ar = [];
    for (let i of li) {
        const last = i.lastElementChild;
        const x = last.textContent.trim();
        const y = Number(x.substring(1));
        i.setAttribute("data-price", y);
        ar.push(i);
    }
```

CODE IMPLEMENTATION

```
function SortProduct() {
    let select = document.getElementById('select');
    let ar = [];
    for (let i of li) {
        const last = i.lastElementChild;
        const x = last.textContent.trim();
        const y = Number(x.substring(1));
        i.setAttribute("data-price", y);
        ar.push(i);
    }

    this.run = () => {
        addevent();
    }

    function addevent() {
        select.onchange = sortingValue;
    }

    function sortingValue() {
        if (this.value === 'Default') {
            restoreDefaultOrder();
        }
        if (this.value === 'LowToHigh') {
            SortElem(field, ar, true);
        }
        if (this.value === 'HighToLow') {
            SortElem(field, ar, false);
        }
    }
```

```
function SortElem(field, arr, asc) {
    while (field.firstChild) {
        field.removeChild(field.firstChild);
    }
    field.append(...arr);
}

function mergeSort(arr, asc) {
    if (arr.length <= 1) return arr;

    const mid = Math.floor(arr.length / 2);
    const left = arr.slice(0, mid);
    const right = arr.slice(mid);

    mergeSort(left, asc);
    mergeSort(right, asc);

    merge(arr, left, right, asc);
}

function merge(arr, left, right, asc) {
    let i = 0,
        j = 0,
        k = 0;
    while (i < left.length && j < right.length) {
        const leftPrice = Number(left[i].getAttribute('data-price'));
        const rightPrice = Number(right[j].getAttribute('data-price'));
        if ((asc && leftPrice <= rightPrice) || (!asc && leftPrice >= rightPrice)) {
            arr[k++] = left[i++];
        } else {
```

CODE IMPLEMENTATION

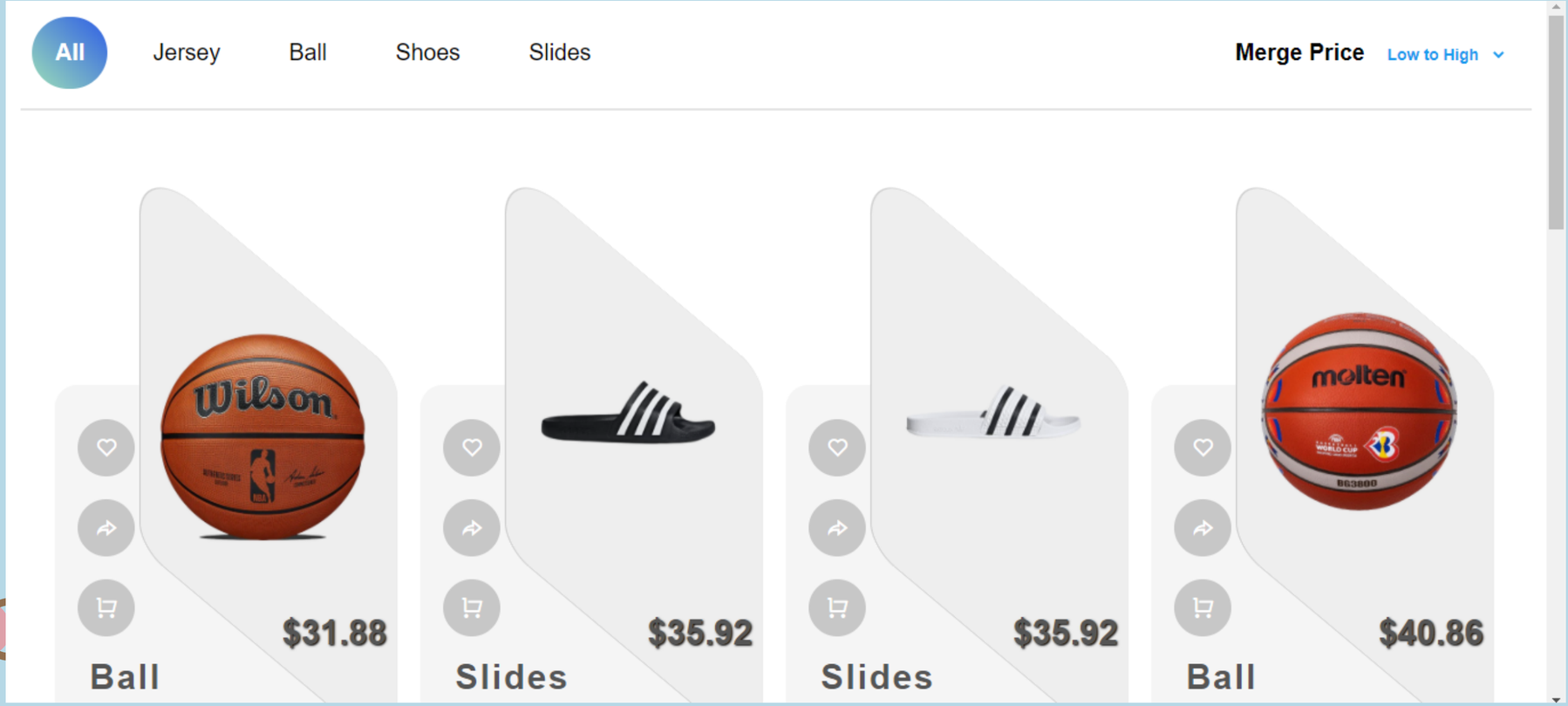
```
                arr[k++] = right[j++];
            }
        }

        while (i < left.length) {
            arr[k++] = left[i++];
        }

        while (j < right.length) {
            arr[k++] = right[j++];
        }
    }

    function restoreDefaultOrder() {
        while (field.firstChild) {
            field.removeChild(field.firstChild);
        }
        originalOrder.forEach(item => field.appendChild(item));
    }
}

new FilterProduct().run();
new SortProduct().run();
})();
```

CODE EXPLANATION.

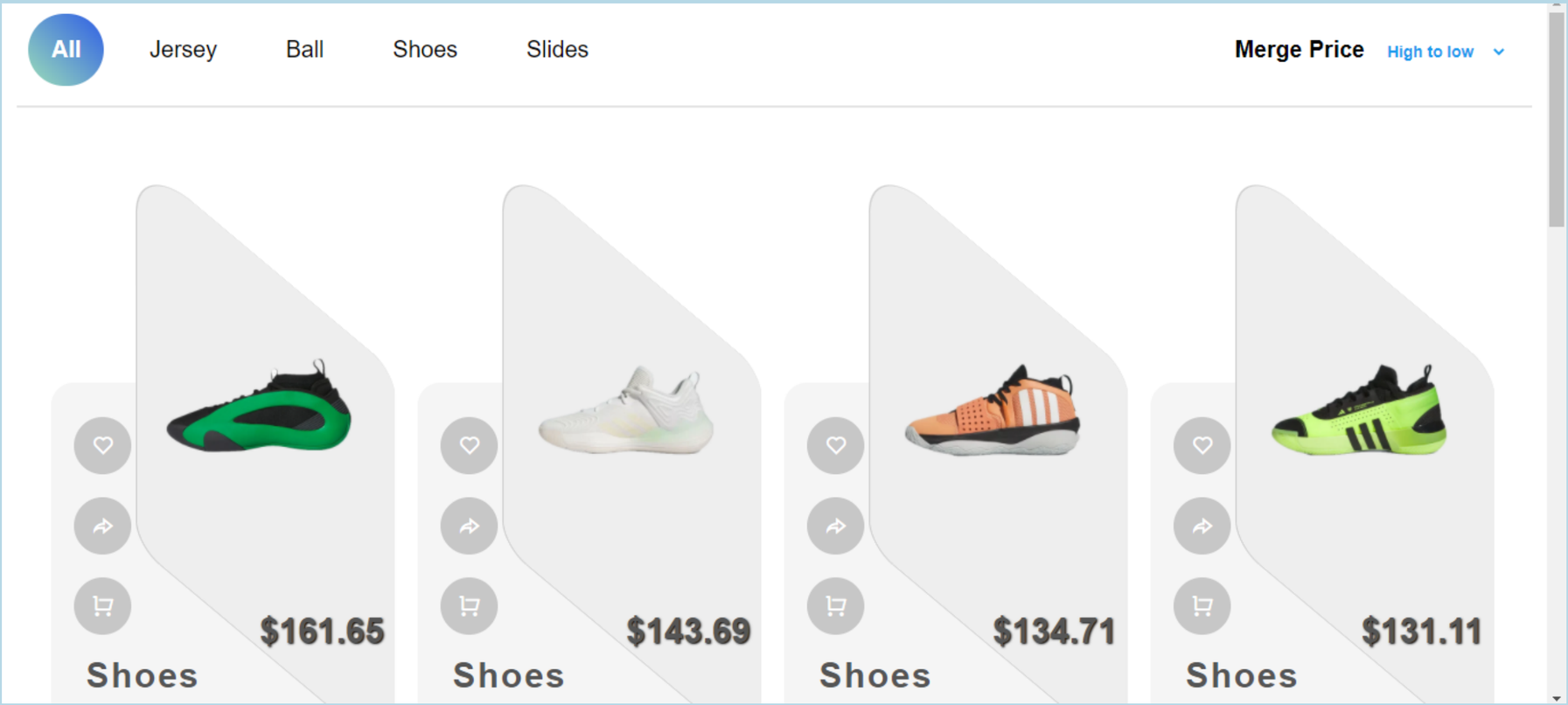
Filtering: The FilterProduct class creates categories for each product. When a category filter is applied, it iterates through each product, hiding products that do not fit the selected category and displaying products that do, or all products if "All" is selected.

Sorting: The SortProduct class sets the prices for each product. When a sorting option is enabled, the products are sorted according to price using merge sort. The mergeSort method breaks an array into smaller arrays, sorts them, and then merges them back together.

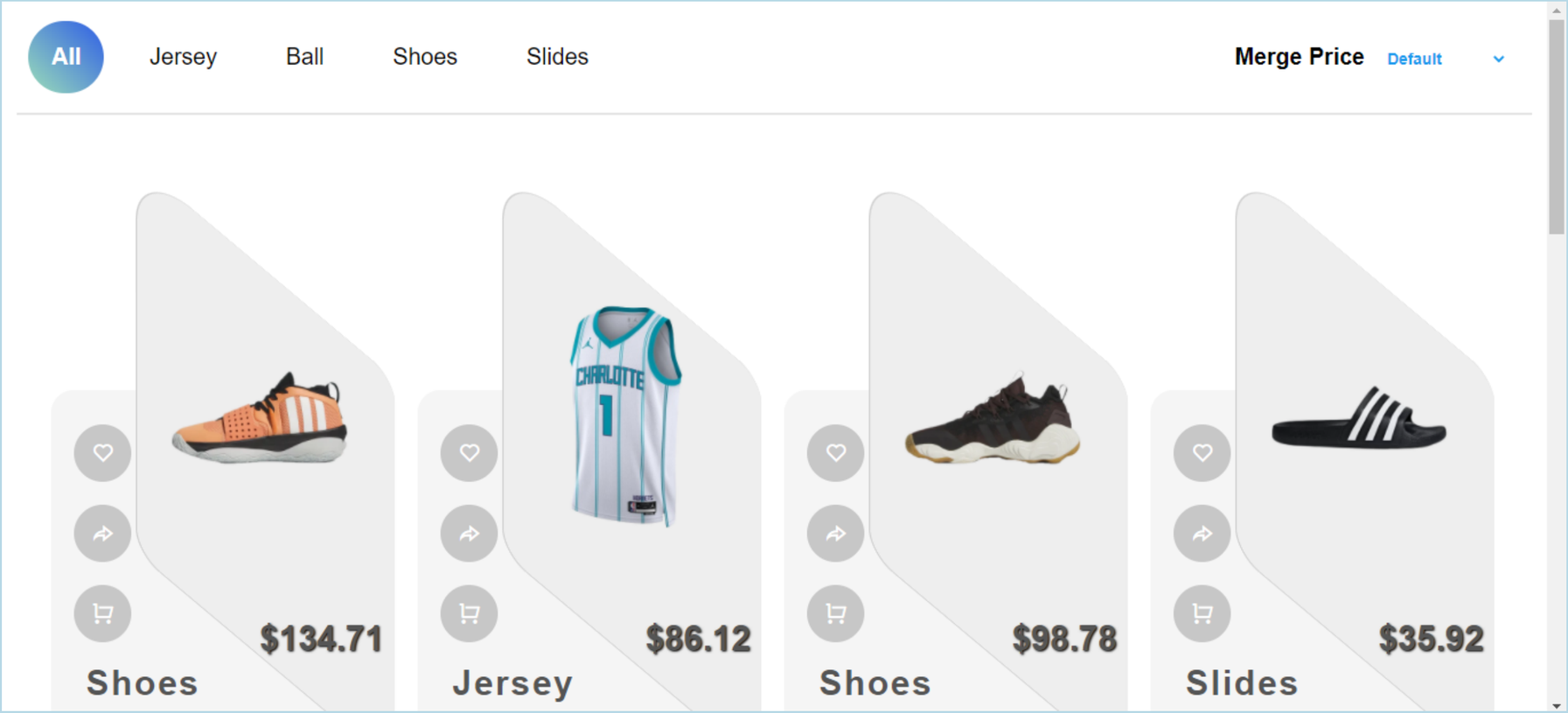
CODE OUTPUT


All
Jersey
Ball
Shoes
Slides
Merge Price
Low to High
Wilson
$31.88
Ball
$35.92
Slides
$35.92
Slides
molten
$40.86
Ball

CODE OUTPUT


All
Jersey
Ball
Shoes
Slides
Merge Price
High to low
$161.65
Shoes
$143.69
Shoes
$134.71
Shoes
$131.11
Shoes

CODE OUTPUT


All
Jersey
Ball
Shoes
Slides
Merge Price
Default
$134.71
Shoes
$86.12
Jersey
$98.78
Shoes
$35.92
Slides

TIME COMPLEXITY

Filtering:
Initialization of categories: O(n), where n is the number of products.
Filtering: O(n), where n is the number of products.

Sorting:
Initialization of prices: O(n), where n is the number of products.
Sorting: O(n log n) using merge sort.

TIME COMPLEXITY

Overall time complexity:

Initialization: O(n)
Filtering: O(n)
Sorting: O(n log n)

LEARNING REFLECTION

During the midterm time, studying the divide and conquer method was both exciting and difficult. Understanding how to break down big problems into smaller, more manageable pieces takes concentrated effort and experience. Despite initial difficulties understanding recursion and merging subproblems, constant practice and seeking clarification from peers and resources enabled me to overcome these challenges. Finally, learning the divide and conquer strategy stressed the need of breaking down challenges systematically, as well as the value of tenacity in mastering difficult ideas.