8. Practical Applications

8.1 Search and Sort Combined

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Student structure
typedef struct {
    int id;
    char name[50];
    float gpa;
} Student;

// Compare functions for sorting
int compareByID(const void *a, const void *b) {
    return ((Student*)a)->id - ((Student*)b)->id;
}

int compareByGPA(const void *a, const void *b) {
    float diff = ((Student*)b)->gpa - ((Student*)a)->gpa;
    return (diff > 0) ? 1 : (diff < 0) ? -1 : 0;
}

int compareByName(const void *a, const void *b) {
    return strcmp(((Student*)a)->name, ((Student*)b)->name);
}

// Binary search by ID
int searchByID(Student students[], int n, int targetID) {
    int left = 0, right = n - 1;

    while (left <= right) {
        int mid = left + (right - left) / 2;
```

```
        if (students[mid].id == targetID) {
            return mid;
        }
        else if (students[mid].id < targetID) {
            left = mid + 1;
        }
        else {
            right = mid - 1;
        }
    }

    return -1;
}

// Linear search by name
int searchByName(Student students[], int n, const char *name) {
    for (int i = 0; i < n; i++) {
        if (strcmp(students[i].name, name) == 0) {
            return i;
        }
    }
    return -1;
}

// Print students
void printStudents(Student students[], int n) {
    printf("\n%-10s %-20s %-8s\n", "ID", "Name", "GPA");
    printf("----------------------------------------\n");
    for (int i = 0; i < n; i++) {
        printf("%-10d %-20s %.2f\n",
               students[i].id, students[i].name, students[i].gpa);
    }
}

int main() {
    Student students[] = {
        {1003, "Alice Johnson", 3.8},
        {1001, "Bob Smith", 3.5},
        {1005, "Charlie Brown", 3.9},
        {1002, "Diana Prince", 3.7},
```

```
        {1004, "Eve Wilson", 3.6}
    };
    int n = sizeof(students) / sizeof(students[0]);

    printf("=== Student Database ===\n");
    printf("\nOriginal Data:");
    printStudents(students, n);

    // Sort by ID
    qsort(students, n, sizeof(Student), compareByID);
    printf("\nSorted by ID:");
    printStudents(students, n);

    // Search by ID
    int targetID = 1003;
    int index = searchByID(students, n, targetID);
    if (index != -1) {
        printf("\nStudent with ID %d: %s (GPA: %.2f)\n",
               targetID, students[index].name, students[index].gpa);
    }

    // Sort by GPA (descending)
    qsort(students, n, sizeof(Student), compareByGPA);
    printf("\nSorted by GPA (highest first):");
    printStudents(students, n);

    // Sort by Name
    qsort(students, n, sizeof(Student), compareByName);
    printf("\nSorted by Name:");
    printStudents(students, n);

    return 0;
}
```

8.2 Finding Kth Largest Element

```
// Partition function (same as Quick Sort)
int partition(int arr[], int low, int high) {
    int pivot = arr[high];
```

```
    int i = low - 1;

    for (int j = low; j < high; j++) {
        if (arr[j] <= pivot) {
            i++;
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }

    int temp = arr[i + 1];
    arr[i + 1] = arr[high];
    arr[high] = temp;

    return i + 1;
}

// Quick Select algorithm - O(n) average time
int findKthLargest(int arr[], int low, int high, int k) {
    if (low == high) {
        return arr[low];
    }

    int pi = partition(arr, low, high);
    int length = high - pi + 1;

    if (length == k) {
        return arr[pi];
    }
    else if (length > k) {
        return findKthLargest(arr, pi + 1, high, k);
    }
    else {
        return findKthLargest(arr, low, pi - 1, k - length);
    }
}

int main() {
    int arr[] = {3, 2, 1, 5, 6, 4};
```

```
    int n = sizeof(arr) / sizeof(arr[0]);
    int k = 2;

    int result = findKthLargest(arr, 0, n - 1, k);
    printf("%dth largest element: %d\n", k, result);
    // Output: 2nd largest element: 5

    return 0;
}
```

8.3 Merge Two Sorted Arrays

```
#include <stdio.h>
#include <stdlib.h>

int* mergeSortedArrays(int arr1[], int n1, int arr2[], int n2) {
    int *result = (int*)malloc((n1 + n2) * sizeof(int));
    int i = 0, j = 0, k = 0;

    // Merge while both arrays have elements
    while (i < n1 && j < n2) {
        if (arr1[i] <= arr2[j]) {
            result[k++] = arr1[i++];
        } else {
            result[k++] = arr2[j++];
        }
    }

    // Copy remaining elements from arr1
    while (i < n1) {
        result[k++] = arr1[i++];
    }

    // Copy remaining elements from arr2
    while (j < n2) {
        result[k++] = arr2[j++];
    }

    return result;
```

```
}

int main() {
    int arr1[] = {1, 3, 5, 7};
    int arr2[] = {2, 4, 6, 8};
    int n1 = sizeof(arr1) / sizeof(arr1[0]);
    int n2 = sizeof(arr2) / sizeof(arr2[0]);

    int *merged = mergeSortedArrays(arr1, n1, arr2, n2);

    printf("Merged array: ");
    for (int i = 0; i < n1 + n2; i++) {
        printf("%d ", merged[i]);
    }
    printf("\n");

    free(merged);
    return 0;
}
```

8.4 Finding Median

```
// Find median of unsorted array
double findMedian(int arr[], int n) {
    // Sort the array first
    quickSort(arr, 0, n - 1);

    // Find median
    if (n % 2 == 0) {
        // Even number of elements - average of two middle
        return (arr[n/2 - 1] + arr[n/2]) / 2.0;
    } else {
        // Odd number of elements - middle element
        return arr[n/2];
    }
}

int main() {
    int arr1[] = {3, 1, 4, 1, 5, 9, 2};
```

```
    int arr2[] = {6, 5, 3, 5, 8, 9};

    printf("Median of arr1: %.1f\n", findMedian(arr1, 7));
    printf("Median of arr2: %.1f\n", findMedian(arr2, 6));

    return 0;
}
```

8.5 Removing Duplicates from Sorted Array

```
int removeDuplicates(int arr[], int n) {
    if (n == 0 || n == 1) {
        return n;
    }

    int j = 0;  // Index for unique elements

    for (int i = 0; i < n - 1; i++) {
        if (arr[i] != arr[i + 1]) {
            arr[j++] = arr[i];
        }
    }

    arr[j++] = arr[n - 1];  // Add last element

    return j;  // New length
}

int main() {
    int arr[] = {1, 1, 2, 2, 2, 3, 4, 4, 5};
    int n = sizeof(arr) / sizeof(arr[0]);

    printf("Original: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    int newLength = removeDuplicates(arr, n);
```

```
    printf("After removing duplicates: ");
    for (int i = 0; i < newLength; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    return 0;
}
```

8.6 Finding Intersection of Two Arrays

```
void findIntersection(int arr1[], int n1, int arr2[], int n2) {
    // Sort both arrays first
    quickSort(arr1, 0, n1 - 1);
    quickSort(arr2, 0, n2 - 1);

    printf("Intersection: ");
    int i = 0, j = 0;

    while (i < n1 && j < n2) {
        if (arr1[i] < arr2[j]) {
            i++;
        }
        else if (arr1[i] > arr2[j]) {
            j++;
        }
        else {
            // Elements are equal
            printf("%d ", arr1[i]);
            i++;
            j++;
        }
    }
    printf("\n");
}

int main() {
    int arr1[] = {1, 3, 4, 5, 7};
```

```
    int arr2[] = {2, 3, 5, 6};

    findIntersection(arr1, 5, arr2, 4);
    // Output: Intersection: 3 5

    return 0;
}
```

8.7 Sorting Strings

```
#include <stdio.h>
#include <string.h>

void sortStrings(char *arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (strcmp(arr[j], arr[j + 1]) > 0) {
                // Swap pointers
                char *temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}

int main() {
    char *fruits[] = {"Banana", "Apple", "Orange", "Mango", "Grape"};
    int n = 5;

    printf("Original: ");
    for (int i = 0; i < n; i++) {
        printf("%s ", fruits[i]);
    }
    printf("\n");

    sortStrings(fruits, n);

    printf("Sorted: ");
    for (int i = 0; i < n; i++) {
```

```
        printf("%s ", fruits[i]);
    }
    printf("\n");

    return 0;
}
```

Revision #1
Created 2025-11-03 02:10:36 UTC by DS
Updated 2025-11-03 02:11:49 UTC by DS