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

int extraMemoryAllocated;

void merge(int pData[], int l, int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 =  r - m;

    int *L = (int*) malloc(n1*sizeof(int));
    int *R = (int*) malloc(n2*sizeof(int));

    for (i = 0; i < n1; i++)
        L[i] = pData[l + i];
    for (j = 0; j < n2; j++)
        R[j] = pData[m + 1+ j];

    i = 0;
    j = 0;
    k = l;
    while (i < n1 && j < n2)
    {
        if (L[i] <= R[j])
        {
            pData[k] = L[i];
            i++;
        }
        else
        {
            pData[k] = R[j];
            j++;
        }
        k++;
    }

    while (i < n1)
    {
        pData[k] = L[i];
        i++;
        k++;
    }
    while (j < n2)
    {
        pData[k] = R[j];
        j++;
        k++;
    }
    free(L);
    free(R);
}

// implement merge sort
// extraMemoryAllocated counts bytes of memory allocated
void mergeSort(int pData[], int l, int r)
{
    if (l < r)
    {
        // middle
        int m = (l+r)/2;

        mergeSort(pData, l, m);
        mergeSort(pData, m+1, r);

        merge(pData, l, m, r);
    }
}

// implement insertion sort
// extraMemoryAllocated counts bytes of memory allocated
void insertionSort(int* pData, int n)
{
    int i, item, j;

    for (i = 1; i < n; i++)
    {
        item = pData[i];

        for (j = (i - 1); j >= 0; j--)
        {
            if (pData[j] > item)
                pData[j + 1] = pData[j];
            else
                break;
        }
        pData[j + 1] = item;
```

```c
    }
}

// implement bubble sort
// extraMemoryAllocated counts bytes of extra memory allocated
void bubbleSort(int* pData, int n)
{
    int i, j, temp;

    for (i = 0; i < n-1; i++)
    {
        for (j = 0; j < n-i-1; j++)
        {
            if (pData[j] > pData[j+1])
            {
                temp = pData[j];
                pData[j] = pData[j+1];
                pData[j+1] = temp;
            }
        }
    }
}

// implement selection sort
// extraMemoryAllocated counts bytes of extra memory allocated
void selectionSort(int* pData, int n)
{
    int i, j, min_idex, temp;

    for (i = 0; i < n-1; i++)
    {
        min_idex = i;
        for (j = i+1; j < n; j++)
            if (pData[j] < pData[min_idex])
                min_idex = j;

        temp = pData[i];
        pData[i] = pData[min_idex];
        pData[min_idex] = temp;
    }
}

// parses input file to an integer array
int parseData(char *inputFileName, int **ppData)
{
    FILE* inFile = fopen(inputFileName,"r");
    int dataSz = 0;
    *ppData = NULL;

    if (inFile)
    {
        fscanf(inFile,"%d\n",&dataSz);
        *ppData = (int *)malloc(sizeof(int) * dataSz);

        int num;

        for(int i = 0; i < dataSz; ++i)
            fscanf(inFile, "%d\n", &((*ppData)[i]));
        // Implement parse data block
    }

    return dataSz;
}

// prints first and last 100 items in the data array
void printArray(int pData[], int dataSz)
{
    int i, sz = dataSz - 100;
    printf("\tData:\n\t");
    for (i=0;i<100;++i)
    {
        printf("%d ",pData[i]);
    }
    printf("\n\t");

    for (i=sz;i<dataSz;++i)
    {
        printf("%d ",pData[i]);
    }
    printf("\n\n");
}

int main(void)
{
    clock_t start, end;
    int i;
    double cpu_time_used;
    char* fileNames[] = {"input1.txt", "input2.txt", "input3.txt"};

    for (i=0;i<3;++i)
```

```c
    {
        int *pDataSrc, *pDataCopy;
        int dataSz = parseData(fileNames[i], &pDataSrc);

        if (dataSz <= 0)
            continue;

        pDataCopy = (int *)malloc(sizeof(int)*dataSz);

        printf("---------------------------\n");
        printf("Dataset Size : %d\n",dataSz);
        printf("---------------------------\n");

        printf("Selection Sort:\n");
        memcpy(pDataCopy, pDataSrc, dataSz*sizeof(int));
        extraMemoryAllocated = 0;
        start = clock();
        selectionSort(pDataCopy, dataSz);
        end = clock();
        cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
        printf("\truntime\t\t\t: %.1lf\n",cpu_time_used);
        printf("\textra memory allocated\t: %d\n",extraMemoryAllocated);
        printArray(pDataCopy, dataSz);

        printf("Insertion Sort:\n");
        memcpy(pDataCopy, pDataSrc, dataSz*sizeof(int));
        extraMemoryAllocated = 0;
        start = clock();
        insertionSort(pDataCopy, dataSz);
        end = clock();
        cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
        printf("\truntime\t\t\t: %.1lf\n",cpu_time_used);
        printf("\textra memory allocated\t: %d\n",extraMemoryAllocated);
        printArray(pDataCopy, dataSz);

        printf("Bubble Sort:\n");
        memcpy(pDataCopy, pDataSrc, dataSz*sizeof(int));
        extraMemoryAllocated = 0;
        start = clock();
        bubbleSort(pDataCopy, dataSz);
        end = clock();
        cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
        printf("\truntime\t\t\t: %.1lf\n",cpu_time_used);
        printf("\textra memory allocated\t: %d\n",extraMemoryAllocated);
        printArray(pDataCopy, dataSz);
        printf("Merge Sort:\n");
        memcpy(pDataCopy, pDataSrc, dataSz*sizeof(int));
        extraMemoryAllocated = 0;
        start = clock();
        mergeSort(pDataCopy, 0, dataSz - 1);
        end = clock();
        cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
        printf("\truntime\t\t\t: %.1lf\n",cpu_time_used);
        printf("\textra memory allocated\t: %d\n",extraMemoryAllocated);
        printArray(pDataCopy, dataSz);

        free(pDataCopy);
        free(pDataSrc);
    }

}
```