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


typedef int (*compare_t)(const void *, const void *);
typedef void (*sort_func_t)(void *, size_t nmemb, size_t size, compare_t);

enum e_workload
{
    ew_num,
    ew_str,
    ew_student,
    ew_count
};

typedef struct _Student
{
    char id[10];
    char name[512];
} Student;

int compare_numbers(const void *lhs, const void *rhs);
int compare_strings(const void *lhs, const void *rhs);
int compare_students(const void *lhs, const void *rhs);

static compare_t compare_funcs[ew_count] = {
    compare_numbers,
    compare_strings,
    compare_students};

enum e_sort_type
{
    est_insertion,
    est_selection,
    est_merge,
    est_count
};

//nmemb = the number of members = the number of elements in the array.
void insertion_sort(void *arr, size_t nmemb, size_t size, compare_t compare);
void selection_sort(void *arr, size_t nmemb, size_t size, compare_t compare);
void merge_sort(void *arr, size_t nmemb, size_t size, compare_t compare);

void *read_data(enum e_workload data_type, size_t data_size, size_t *count);

static sort_func_t sort_funcs[est_count] = {
    insertion_sort,
    selection_sort,
    merge_sort};

static const char *sort_funcs_name[est_count] = {
    "Insertion sort",
    "Selection sort",
    "Merge sort"};

int main(int argc, char **argv)
{

    FILE* pFile=fopen("TIMEEEE.txt", "a");


    if (argc != 3)
    {
        fprintf(stderr, "Usage: ./sort (num | str | student) (insertion |
selection | merge) < input_filepath > output_filepath\n");
        exit(-1);
    }

    enum e_workload workload_type = ew_num;

    size_t size = sizeof(int);

    if (strcmp(argv[1], "num") == 0)
    {
        workload_type = ew_num;
        size = sizeof(int);
    }
    else if (strcmp(argv[1], "str") == 0)
    {
        workload_type = ew_str;
        size = sizeof(char);
    }
    else if (strcmp(argv[1], "student") == 0)
    {
        workload_type = ew_student;
        size = sizeof(Student);
    }

    sort_func_t sort_func;
    if (strcmp(argv[2], "insertion") == 0)
    {
        sort_func = insertion_sort;
    }
    else if (strcmp(argv[2], "selection") == 0)
    {
        sort_func = selection_sort;
    }
    else if (strcmp(argv[2], "merge") == 0)
    {
        sort_func = merge_sort;
    }

    // The number of data
    size_t count = 0;

    void *arr = read_data(workload_type, size, &count);

    struct timeval start;
    gettimeofday(&start, NULL);


    sort_func(arr, count, size, compare_funcs[workload_type]);

    struct timeval end;
    gettimeofday(&end, NULL);

    fprintf(pFile, " %lf\n", ((end.tv_sec + end.tv_usec * 0.000001) -
(start.tv_sec + start.tv_usec * 0.000001)));


    for (int i = 0; i < (int)count; i++){

        switch (workload_type)
        {
        case ew_num:
            fprintf(stdout,"%d\n", ((int*)arr)[i]);
            break;

        case ew_str:
            fprintf(stdout, "%c\n", ((char*)arr)[i]);
            break;

        case ew_student:
            fprintf(stdout, "%s %s\n", ((Student*)arr)[i].id,
((Student*)arr)[i].name);
            break;

        default:
            break;
        }

    }
```

```c
    free(arr);
    return 0;
}

// Scenario 1: Number sorting
int compare_numbers(const void* lhs, const void* rhs)
{
    int* lhs_int = (int*)lhs;
    int* rhs_int = (int*)rhs;
    if (*lhs_int > *rhs_int)return 0;  //오른이 작으면 0
    if (*lhs_int == *rhs_int)return 1;  //       같으면 1
    if (*lhs_int < *rhs_int)return 2;  //왼쪽이 작으면 2

}

// Scenario 2: Single digit string sorting - using ASCII
int compare_strings(const void* lhs, const void* rhs)
{
    char* lhs_char = (char*)lhs;
    char* rhs_char = (char*)rhs;
    char lhs_charr;
    char rhs_charr;

    if(*lhs_char>='a'&&*lhs_char<='z')lhs_charr= *lhs_char-'a'+'A';
    else lhs_charr=*lhs_char;
    if(*rhs_char>='a'&&*rhs_char<='z')rhs_charr= *rhs_char-'a'+'A';
    else rhs_charr=*rhs_char;

    if (lhs_charr > rhs_charr)return 0;  //오른이 작으면 0
    if (lhs_charr== rhs_charr){
        if (*lhs_char < *rhs_char)return 0;  //오른이 작으면 0
        if (*lhs_char == *rhs_char)return 1; //       같으면 1
        if (*lhs_char > *rhs_char)return 2;  //왼쪽이 작으면 2
    }
    if (lhs_charr < rhs_charr)return 2;  //왼쪽이 작으면 2

}

// Scenario 3: Struct sorting. Sort by ID not name.
int compare_students(const void* lhs, const void* rhs)
{
    Student* lhs_Stu = (Student*)lhs;
    Student* rhs_Stu = (Student*)rhs;

    int i=0;
    while(1){
        if (((*lhs_Stu).id)[i] == 0||((*rhs_Stu).id)[i] == 0)return 1;
        if (((*lhs_Stu).id)[i] > ((*rhs_Stu).id)[i])return 0;  //오른이 작으면
0
        if (((*lhs_Stu).id)[i] < ((*rhs_Stu).id)[i])return 2;  //왼쪽이 작으면
2
        if (((*lhs_Stu).id)[i] == ((*rhs_Stu).id)[i]){
            if(i<9)i++;
            else if(i>=9){
                return 1;//같으면 1
            }
        }
    }

}

int compare_students2(const void* lhs, const void* rhs)
{
    Student* lhs_Stu = (Student*)lhs;
    Student* rhs_Stu = (Student*)rhs;

    int i=0;
    int lhs_id=0,rhs_id=0;
    for(int i=0;i<10 && (*lhs_Stu).id[i]!=0;i++){
        lhs_id=10*lhs_id + (*lhs_Stu).id[i];
    }
    for(int i=0;i<10 && (*rhs_Stu).id[i]==0;i++){
        rhs_id=10*rhs_id + (*rhs_Stu).id[i];
    }

    if (lhs_id > rhs_id)return 0;  //오른이 작으면 0
    if (lhs_id == rhs_id)return 1; //       같으면 1
    if (lhs_id < rhs_id)return 2;  //왼쪽이 작으면 2

}

void insertion_sort(void* arr, size_t nmemb, size_t size, compare_t compare){

char* arr_1= (char*) arr;

   for (int i = 1; i < nmemb; i++) {
      for(int j = i - 1; j >= 0 && compare(arr_1 + size * j, arr_1 + size * (j
+ 1)) == 0; j--){

         void* temp = malloc(size);
         memcpy(temp, (arr_1 + size * (j + 1)), size);
         memcpy((arr_1 + size * (j + 1)), (arr_1 + size * j), size);
         memcpy((arr_1 + size * j), temp, size);
         free(temp);
```

```
            }
        }


    }

    void selection_sort(void *arr, size_t nmemb, size_t size, compare_t compare)
    {

        int i, j, min_num = 0;
        char* arr_1= (char*) arr;

        for (i = 0;i < nmemb-1;i++) {
            min_num = i;
            for (j = i+1;j < nmemb;j++) {
                if (compare(arr_1+size*min_num, arr_1+size*j) != 2) {
                    min_num = j;
                }
            }

            void* temp = malloc(size);
            memcpy(temp, (arr_1 + size * (i)), size);
            memcpy((arr_1 + size * (i)), (arr_1 + size * min_num), size);
            memcpy((arr_1 + size * min_num), temp, size);
            free(temp);

        }
    }


    void conqure(void* arr, int start, int mid, int end,size_t size,compare_t
    compare){
        int i=0, j=0, k=0;
        int arr_size= end-start;

        char* arr_1 = (char*)arr;
        void* new_arr_void = malloc(size * (arr_size+1));
        char* new_arr=(char*)new_arr_void;

        while(start+i<mid&&mid+j<end){
            if(compare(arr_1+size*(start+i),arr_1+size*(mid+j))==0){
                memcpy(new_arr+size*k,arr_1+size*(mid+j),size);
                j++;
            }
            else{
                memcpy(new_arr+size*k,arr_1+size*(start+i),size);
                i++;
            }
```

```c
        k++;
    }

    if(mid+j<end){
        memcpy(new_arr+size*k,arr_1+size*(mid+j),size*(arr_size-k));
    }

    if(start+i<mid){
        memcpy(new_arr+size*k,arr_1+size*(start+i),size*(arr_size-k));
    }

    memcpy(arr_1+size*start,new_arr,size*arr_size);

    free(new_arr);
}

void merge(void *arr, int start, int arr_size,size_t size,compare_t compare){

    if(arr_size!=1){

        merge(arr, start, arr_size/2,size,compare);
        merge(arr, start+arr_size/2,arr_size-arr_size/2,size,compare);
        conqure(arr,start,start+arr_size/2,start+arr_size,size,compare);
    }

}
void merge_sort(void *arr, size_t nmemb, size_t size, compare_t compare)
{
    merge(arr, 0,nmemb,size,compare);

}

int count_num_line(FILE *fp)
{
    int count = 0;
    char c;
    while ((c = fgetc(fp)) != EOF)
    {
        if (c == '\n')
            count++;
    }

    rewind(fp);

    return count;
}

void *read_data(enum e_workload data_type, size_t data_size, size_t *count)
```

```
{
    *count = count_num_line(stdin);

    void *arr = malloc(data_size * (*count));

    for (size_t i = 0; i <*count; i++)
    {
        int result;
        char str_temp[4];

        switch (data_type)
        {
        case ew_num:
            result = fscanf(stdin, "%d", &(((int *)arr)[i]));

            break;

        case ew_str:
            result = fscanf(stdin, "%s", str_temp);
            ((char *)arr)[i] = str_temp[0];

            break;

        case ew_student:
            result = fscanf(stdin, "%s %s", ((Student *)arr)[i].id, ((Student
*)arr)[i].name);
            break;

        default:
            break;
        }
        if(result<=0){
            //break;
        }

    }

    return arr;

}
```