

```
typedef struct Page
{
    int index;
    int array[PAGE_SIZE];
    int load_time;
    int last_reference_time;
    int R_bit;
    int M_bit;
    int second_chance;
}Page;

typedef struct pagetable {
    Page pages[NUMBER_OF_PAGE];
}pagetable;

typedef struct my_program {
    pagetable table;
}my_program;

#endif
```

```
#define VIRTUAL_ADDRESS_SPACE 64 //2^6
#define PAGE_SIZE 16 //2^4
#define NUMBER_OF_PAGE 4 //2^2
#define DISK_ARRAY_SIZE 128
```

I have these structs, My program holds a page table and it has a number of pages in it, then each page has integer content, disk index, load time last reference time, R-M bits and second chance bit to indicate second chance

Firstly I created arrays and put the array to the page table, but array length was high such that I can't hold the entire array in my pages. So I decided to start from index 0 of array, and copy the elements as soon as I can. In this case, it is half of the entire array. So, My page table can hold half of the entire array.

```
int disk_array[DISK_ARRAY_SIZE];
int insertion_sort_array[DISK_ARRAY_SIZE];
int bubble_sort_array[DISK_ARRAY_SIZE];
int quick_sort_array[DISK_ARRAY_SIZE];
for(int i=0;i<DISK_ARRAY_SIZE;i++) {
    disk_array[i]=DISK_ARRAY_SIZE-i;
    insertion_sort_array[i]=disk_array[i];
    bubble_sort_array[i]=disk_array[i];
    quick_sort_array[i]=disk_array[i];
    char buf[50];
    toChar(buf,(disk_array[i]));
    printf(buf);
    printf(" ");
}
```

```
for(int i=0;i<NUMBER_OF_PAGE;i++) {
    p.table.pages[i].last_reference_time=0;
    p.table.pages[i].load_time=i;
    p.table.pages[i].M_bit=0;
    p.table.pages[i].R_bit=0;
    p.table.pages[i].second_chance=0;
    p.table.pages[i].index=i;
    loaded_page++;
    for(int j=0;j<PAGE_SIZE;j++) {
        p.table.pages[i].array[j]=disk_array[i*PAGE_SIZE+j];
    }
}
```

Bubble Sort:

```
void bubbleSort(my_program*p,int arr[], int n)
{
    int i, j;
    for (i = 0; i < n - 1; i++) {
        for (j = 0; j < n - i - 1; j++) {
            int* arrj=reference_memory(p,arr,j);
            int* arrjp=reference_memory(p,arr,j+1);
            if ((*arrj)>(*arrjp)) {
                swap(arrj,arrjp);
            }
        }
    }
}
```

My Operating System [Çalışıyor] - Oracle VM VirtualBox

Dosya Makine Görünüm Giriş Aygıtlar Yardım

128 127 126 125 124 123 122 121 120 119 118 117 116 115 114 113 112 111 110 109
108 107 106 105 104 103 102 101 100 99 98 97 96 95 94 93 92 91 90 89 88 87 86 85
84 83 82 81 80 79 78 77 76 75 74 73 72 71 70 69 68 67 66 65 64 63 62 61 60 59 58
57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3
2 1
SECOND CHANCE ALGORITHM
BUBBLE OPERATION
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57
58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83
84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107
108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127
128
TOTAL HIT:16256
TOTAL MISS:326
LOADED PAGE:330

Quick Sort:

```
int partition (my_program*p,int arr[], int low, int high){
    int pivot=(*reference_memory(p,arr,high));
    int i = (low - 1);
    for (int j = low; j <= high - 1; j++){
        if (*reference_memory(p,arr,j) < pivot){
            i++;
            swap(reference_memory(p,arr,i), reference_memory(p,arr,j));
        }
    }
    swap(reference_memory(p,arr,i+1), reference_memory(p,arr,high));
    return (i + 1);
}

void quickSort(my_program*p,int arr[], int low, int high){
    if (low < high){
        int pi = partition(p,arr, low, high);
        quickSort(p,arr, low, pi - 1);
        quickSort(p,arr, pi + 1, high);
    }
}
```

```
My Operating System [Çalışıyor] - Oracle VM VirtualBox
Dosya Makine Görünüm Giriş Aygıtlar Yardım
128 127 126 125 124 123 122 121 120 119 118 117 116 115 114 113 112 111 110 109
108 107 106 105 104 103 102 101 100 99 98 97 96 95 94 93 92 91 90 89 88 87 86 85
84 83 82 81 80 79 78 77 76 75 74 73 72 71 70 69 68 67 66 65 64 63 62 61 60 59 58
57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3
2 1
SECOND CHANCE ALGORITHM
QUICK OPERATION
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57
58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83
84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107
108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127
128
TOTAL HIT:16259
TOTAL MISS:369
LOADED PAGE:373
WRITTEN PAGE:373
```

Insertion sort:

```
void insertionSort(my_program*p,int arr[], int n){
    int i, key, j;
    for (i = 1; i < n; i++){
        key=(*reference_memory(p,arr,i));
        j = i - 1;
        while (j >= 0 && (*reference_memory(p,arr,j)) > key){
            (*reference_memory(p,arr,j+1))=(*reference_memory(p,arr,j));
            j = j - 1;
        }
        *reference_memory(p,arr,j+1)=key;
    }
}
```

```
My Operating System [Çalışıyor] - Oracle VM VirtualBox
Dosya Makine Görünüm Giriş Aygıtlar Yardım
128 127 126 125 124 123 122 121 120 119 118 117 116 115 114 113 112 111 110 109
108 107 106 105 104 103 102 101 100 99 98 97 96 95 94 93 92 91 90 89 88 87 86 85
84 83 82 81 80 79 78 77 76 75 74 73 72 71 70 69 68 67 66 65 64 63 62 61 60 59 58
57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3
2 1
SECOND CHANCE ALGORITHM
INSERTION OPERATION
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57
58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83
84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107
108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127
128
TOTAL HIT:24638
TOTAL MISS:349
LOADED PAGE:353
```

Fifo replacement

```
void fifo_replacement(my_program*p,int arr_index,int*arr) {
    int min_load_time=current_clock;
    int page_index=0;
    int page_i=0;
    for(int i=0;i<NUMBER_OF_PAGE;i++) {
        if(p->table.pages[i].load_time<min_load_time) {
            page_i=i;
            page_index=p->table.pages[i].index;
            min_load_time=p->table.pages[i].load_time;
        }
    }
    int needed_start_index=(arr_index/PAGE_SIZE)*PAGE_SIZE;
    for(int i=needed_start_index;i<(needed_start_index+PAGE_SIZE);i++) {
        p->table.pages[page_i].array[i%PAGE_SIZE]=arr[i];
    }
    p->table.pages[page_i].index=needed_start_index/PAGE_SIZE;
    p->table.pages[page_i].last_reference_time=current_clock;
    p->table.pages[page_i].load_time=current_clock++;
    p->table.pages[page_i].M_bit=0;
    p->table.pages[page_i].R_bit=0;
    p->table.pages[page_i].second_chance=0;
}
```

Looks for minimum load time

Second chance replacement:

```
void second_chance_replacement(my_program*p,int arr_index,int*arr) {
    int page_index=0;
    int page_i=0;
    int flag=0;
    while(!flag) {
        for(int i=0;i<NUMBER_OF_PAGE;i++) {
            if(p->table.pages[i].second_chance==0) {
                page_i=i;
                page_index=p->table.pages[i].index;
                flag=1;
                break;
            }
            else if(p->table.pages[i].second_chance==1) {
                p->table.pages[i].second_chance=0;
            }
        }
    }
    int needed_start_index=(arr_index/PAGE_SIZE)*PAGE_SIZE;
    for(int i=needed_start_index;i<(needed_start_index+PAGE_SIZE);i++) {
        p->table.pages[page_i].array[i%PAGE_SIZE]=arr[i];
    }
    p->table.pages[page_i].index=needed_start_index/PAGE_SIZE;
    p->table.pages[page_i].last_reference_time=
    p->table.pages[page_i].load_time=current_clock++;
    p->table.pages[page_i].M_bit=0;
    p->table.pages[page_i].R_bit=0;
    p->table.pages[page_i].second_chance=0;
}
```

Looks for second chance - R bits, if R bit is 0, replace it, if R bit is 1, make it 0

Least recently used

Looks for last reference time, replaces minimum one

```
void least_recently_used(my_program*p,int arr_index,int*arr) {
    int min_ref_time=current_clock;
    int page_index=0;
    int page_i=0;
    for(int i=0;i<NUMBER_OF_PAGE;i++) {
        if(p->table.pages[i].last_reference_time<min_ref_time) {
            page_i=i;
            page_index=p->table.pages[i].index;
            min_ref_time=p->table.pages[i].load_time;
        }
    }
    int needed_start_index=(arr_index/PAGE_SIZE)*PAGE_SIZE;
    for(int i=needed_start_index;i<(needed_start_index+PAGE_SIZE);i++) {
        p->table.pages[page_i].array[i%PAGE_SIZE]=arr[i];
    }
    p->table.pages[page_i].index=needed_start_index/PAGE_SIZE;
    p->table.pages[page_i].last_reference_time=current_clock;
    p->table.pages[page_i].load_time=current_clock++;
    p->table.pages[page_i].M_bit=0;
    p->table.pages[page_i].R_bit=0;
    p->table.pages[page_i].second_chance=0;
}
```

Referencing a memory

```
int* reference_memory(my_program*p,int arr[], int index) {
    int needed_page_index=(index/PAGE_SIZE);
    for(int i=0;i<NUMBER_OF_PAGE;i++) {
        if(p->table.pages[i].index==needed_page_index) {
            p->table.pages[i].last_reference_time=current_clock++;
            p->table.pages[i].R_bit=1;
            p->table.pages[i].second_chance=1;
            print_hit();
            return &(p->table.pages[i].array[index%PAGE_SIZE]);
        }
    }
    print_miss();
    replace_page(p,index,arr);
    return reference_memory(p,arr,index);
}
```

Replacing a page

```
void replace_page(my_program*p,int arr_index,int*arr) {
    if(fifo_flag==1) {
        fifo_replacement(p,arr_index,arr);
    }
    else if(lru_flag==1) {
        least_recently_used(p,arr_index,arr);
    }
    else if(second_chance_flag==1) {
        second_chance_replacement(p,arr_index,arr);
    }
    else {
        printf("ERROR DETECT TO REPLACEMENT ALGO\n");
    }
    loaded_page++;
    written_page++;
    return;
}
```

Writing back page to the disk

```
void write_back_to_disk(my_program*p,int*array) {
    for(int i=0;i<NUMBER_OF_PAGE;i++) {
        written_page++;
        for(int j=0;j<PAGE_SIZE;j++) {
            array[p->table.pages[i].index*PAGE_SIZE+j]=p->table.pages[i].array[j];
        }
    }
}
```