



OS HW2

Zafer ALTAY

Structures

```

197
198     int arrayForSorting[40]={40,39,38,37,36,35,34,33,32,31,
199                             30,29,28,27,26,25,24,23,22,21,
200                             20,19,18,17,16,15,14,13,12,11,
201                             10,9,8,7,6,5,4,3,2,1};
202
203     const int arraySize=40;
204     const int memorySize=10;
205     int replacementCounter=0;
206     int hitCounter=0;
207     int missCounter=0;
208     int fifolist[memorySize];
209     int pmCounter=0; //Memoryde ki eleman sayısı, memorysize'dan küçükse direkt oraya eklenir.
210
211
212     struct{
213         int val;
214         int rBit;
215         int validBit;
216         int counter; //for LRU
217         int physicalFrame;
218     }typedef PageEntry;
219
220
221     struct
222     {
223         PageEntry pagetable[arraySize];
224     }typedef VirtualMemory;
225     VirtualMemory vm;
226
227     struct
228     {
229         int pages[memorySize];
230     }typedef PhysicalMemory;
231     PhysicalMemory pm;
232
233     struct{
234         int diskPages[arraySize];
235     }typedef Disk;
236     Disk disk;
237

```

Here some values are used for demo. Virtual memory holds page table that consist of page entry and physical memory only hold page numbers. Page information is kept in the page table. If an operation is applied on the page numbers kept in the Memory, the page table is updated.

Algorithms

FirstInFirstOut

```

359 void firstInFirstOut(int newpage){
360                                     //Fifolisttin ilk elemanındaki page'i memoryde bulur kaldırır.
361                                     //Yeni gelecek page'i memory'de onun yerine ekler.
362                                     //Fifo listi günceller.
363     //SİLİNECEK
364     printf("Mem: ");
365     printMemory();
366     printf("\n");
367     printf("Fif: ");
368     printFifoList();
369     printf("\n");
370     /*for(int j=0;j<80000;j++){
371         for (int k = 0; k < 20000; k++)
372         {
373             //SLEEP
374         }
375     }
376     */
377     int i=0;
378     int temp=fifolist[i];
379     int vmIndex=findInVM(temp);
380     vm.pagetable[vmIndex].physicalFrame=-1;
381     vm.pagetable[vmIndex].validBit=0;
382     for (i = 0; i < memorySize-1; i++)
383     {
384         fifolist[i]=fifolist[i+1];
385     }
386     fifolist[memorySize-1]=newpage;
387     for (int i = 0; i < memorySize; ++i)
388     {
389         if (pm.pages[i]==temp)
390         {
391             pm.pages[i]=newpage;
392             int vmIndex=findInVM(newpage);
393             vm.pagetable[vmIndex].physicalFrame=i;
394             vm.pagetable[vmIndex].rBit=1;
395             vm.pagetable[vmIndex].validBit=1;
396             replacementCounter++;
397         }
398     }
399 }
400
401

```

Here I am backing up the first element of fifolist first. Because this is the page to be removed. Then I find this page in pagetable and memory and add the new page in its place. I am updating the page table according to these operations. Then I add the new page to the end of the fifolist.

Second Change

```
void secondC(int newpage){  
  
    int i=0;  
    while(1){  
        int temp=fifolist[0];           //Fifo listteki ilk elemanın rbiti 0'sa fifo fonksiyonunu çağırır,1 ise 0 yapıp en sona atar kaydırır.  
        int vmIndex=findInVM(temp);      //0 olan fifoda kaldırılır. Memory'e eklenir. Page table güncellenir.Fifo list güncellenir.  
  
        printf("Fif: ");  
        printFifoList();  
        printf("\n");  
  
        if (vm.pagetable[vmIndex].rBit==0) //  
        {  
            firstInFirstOut(newpage);  
            return;  
        }  
        else{  
            vm.pagetable[vmIndex].rBit=0;  
            for (int i = 0; i < memorySize-1; i++)  
            {  
                fifolist[i]=fifolist[i+1];  
            }  
            fifolist[memorySize-1]=temp;  
        }  
    }  
}
```

Here I find the first element of our fifo list in pagetable. If the Rbit is 1, I set the rbit to 0 and put it at the end of the list. And I scroll the list 1 time. So I'm looking at the next element. If rbit is 0, I send the new element to fifo, the old element is removed.

LRU

```

3 void LRUSoftware(int newpage){           //Minimum counter'a sahip page'i arar. Onu kaldırır. Yeni gelen page'i onun yerine ekler.
4
5     int min=99999;                       //Page table'ı günceller.
6     int index=-1;
7     int pageNum=-1;
8
9     for (int i = 0; i < memorySize; i++)
10    {
11        int xx=pm.pages[i];
12        int vmIndex=findInVM(xx);
13        if (min>vm.pagetable[vmIndex].counter)
14        {
15            min=vm.pagetable[vmIndex].counter;
16            index=i;
17            pageNum=xx;
18        }
19    }
20    //SİLİNECEK
21    printf("Mem: ");
22    printMemory();
23    printf("\n");
24    printf("Fif: ");
25    printFifoList();
26    printf("\n");
27    /* for(int j=0;j<80000;j++){
28        for (int k = 0; k < 20000; k++)
29        {
30            //SLEEP
31        }
32    }
33    */
34    int vmIndex=findInVM(pageNum);
35    vm.pagetable[vmIndex].physicalFrame=-1;
36    int vmIndex2=findInVM(newpage);
37    vm.pagetable[vmIndex2].physicalFrame=index;
38    pm.pages[index]=newpage;
39    replacementCounter++;
40 }

```

Here I find the page with the lowest counter in memory. I remove it and add the new page. Then I update the page table.

SORTS

Bubble Sort

```

473 //-----SORT ALGORITHM-----
474 void bubbleSort(int arr[], int n,int algorithm){          // bubble sort function
475
476     for(int i = 0; i < n; i++){
477         for(int j = 0; j < n-i-1; j++){                  // loop until n-i-1(further is unnecessary)
478
479             if (!findInMemory(arr[j])) //Page replacement for arr[j] + Eğer memory dolu değilse direkt memory'e ekleme yapılır.
480             {
481                 int vmIndex=findInVM(arr[j]);
482                 int diskIndex=findInDisk(arr[j]);
483                 if (pmCounter<memorySize)
484                 {
485                     pm.pages[pmCounter]=disk.diskPages[diskIndex]; //get page from disk
486                     vm.pagetable[vmIndex].physicalFrame=pmCounter;
487                     vm.pagetable[vmIndex].rBit=1;
488                     vm.pagetable[vmIndex].validBit=1;
489                     fifoList[pmCounter]=disk.diskPages[diskIndex];
490                     pmCounter++;
491                 }
492             }
493             else{                                         //Eğer memory doluysa algoritmaya göre page replacement yapılır.
494                 if (algorithm==1)
495                 {
496                     firstInFirstOut(arr[j]);
497                 }
498                 else if(algorithm==2){
499                     secondC(arr[j]);
500                 }
501                 else{
502                     LRUsoftware(arr[j]);
503                 }
504             }
505         }
506     }
507     if (!findInMemory(arr[j+1])) //Page Replacement for arr[j+1] + Eğer memory dolu değilse direkt memory'e ekleme yapılır.
508     {
509         int vmIndex=findInVM(arr[j+1]);
510         int diskIndex=findInDisk(arr[j+1]);
511         if (pmCounter<memorySize)
512         {
513             pm.pages[pmCounter]=disk.diskPages[diskIndex]; //get page from disk
514             vm.pagetable[vmIndex].physicalFrame=pmCounter;
515             vm.pagetable[vmIndex].rBit=1;
516             vm.pagetable[vmIndex].validBit=1;
517             fifoList[pmCounter]=disk.diskPages[diskIndex];
518             pmCounter++;
519         }
520     }
521     else{
522         if (algorithm==1)
523         {
524             firstInFirstOut(arr[j+1]);
525         }
526         else if(algorithm==2){
527             secondC(arr[j+1]);
528         }
529         else{
530             LRUsoftware(arr[j+1]);
531         }
532     }
533 }
534
535 if( arr[j] > arr[j+1]){
536     int temp = arr[j];
537     arr[j] = arr[j+1];
538     arr[j+1] = temp;
539 }
540 }
541 }
542 }

```

Classic bubble sort algorithm is applied. It has been checked whether the elements of the array to which the sort algorithm will be applied are in memory or not. We do not need to have all of them. Because we are paging here. Thus, it is sufficient for us to have only the element used at that moment. If the memory is not completely full at that moment, it is added directly to the memory. If the memory is full and the page is not in memory, page replacement is performed.

Insertion Sort

```

552 void insertionSort(int arr[], int n,int algorithm)
553 {
554     //-----
555     int i, key, j;
556     for (i = 1; i < n; i++)
557     {
558         //-----
559         if (!findInMemory(arr[i])) //Page replacement for arr[j] + Eğer memory dolu değilse direkt memory'e ekleme yapılır.
560         {
561             int vmIndex=findInVM(arr[i]);
562             int diskIndex=findInDisk(arr[i]);
563             if (pmCounter<memorySize)
564             {
565                 pm.pages[pmCounter]=disk.diskPages[diskIndex]; //get page from disk
566                 vm.pagetable[vmIndex].physicalFrame=pmCounter;
567                 vm.pagetable[vmIndex].rBit=1;
568                 vm.pagetable[vmIndex].validBit=1;
569                 ffolist[pmCounter]=disk.diskPages[diskIndex];
570                 pmCounter++;
571             }
572             else{
573                 //Eğer memory doluysa algoritmaya göre page replacement yapılır.
574                 if (algorithm==1)
575                 {
576                     firstInFirstOut(arr[i]);
577                 }
578                 else if(algorithm==2){
579                     secondC(arr[i]);
580                 }
581                 else{
582                     LRUSoftware(arr[i]);
583                 }
584             }
585         }
586     }
587     key = arr[i];
588     j = i - 1;
589     //-----
590     if (!findInMemory(arr[j])) //Page replacement for arr[j] + Eğer memory dolu değilse direkt memory'e ekleme yapılır.
591     {
592         int vmIndex=findInVM(arr[j]);
593         int diskIndex=findInDisk(arr[j]);
594         if (pmCounter<memorySize)
595         {
596             pm.pages[pmCounter]=disk.diskPages[diskIndex]; //get page from disk
597             vm.pagetable[vmIndex].physicalFrame=pmCounter;
598             vm.pagetable[vmIndex].rBit=1;
599             vm.pagetable[vmIndex].validBit=1;
600             ffolist[pmCounter]=disk.diskPages[diskIndex];
601             pmCounter++;
602         }
603         else{
604             //Eğer memory doluysa algoritmaya göre page replacement yapılır.
605             if (algorithm==1)
606             {
607                 firstInFirstOut(arr[j]);
608             }
609         }
610     }

```

```

711         else if(algorithm==2){
712             secondC(arr[j]);
713         }
714         else{
715             LRUSoftware(arr[j]);
716         }
717     }
718     // Move elements of arr[0..i-1],
719     // that are greater than key, to one
720     // position ahead of their
721     // current position
722     while (j >= 0 && arr[j] > key)
723     {
724         //-----
725         if (!findInMemory(arr[j+1])) //Page replacement for arr[j] + Eğer memory dolu değilse direkt memory'e ekleme yapılır.
726         {
727             int vmIndex=findInVM(arr[j+1]);
728             int diskIndex=findInDisk(arr[j+1]);
729             if (pmCounter<memorySize)
730             {
731                 pm.pages[pmCounter]=disk.diskPages[diskIndex]; //get page from disk
732                 vm.pagetable[vmIndex].physicalFrame=pmCounter;
733                 vm.pagetable[vmIndex].rBit=1;
734                 vm.pagetable[vmIndex].validBit=1;
735                 ffolist[pmCounter]=disk.diskPages[diskIndex];
736                 pmCounter++;
737             }
738             else{
739                 //Eğer memory doluysa algoritmaya göre page replacement yapılır.
740                 if (algorithm==1)
741                 {
742                     firstInFirstOut(arr[j+1]);
743                 }
744                 else if(algorithm==2){
745                     secondC(arr[j+1]);
746                 }
747                 else{
748                     LRUSoftware(arr[j+1]);
749                 }
750             }
751         }
752     }
753     if (!findInMemory(arr[j])) //Page replacement for arr[j] + Eğer memory dolu değilse direkt memory'e ekleme yapılır.
754     {
755         int vmIndex=findInVM(arr[j]);
756         int diskIndex=findInDisk(arr[j]);
757         if (pmCounter<memorySize)
758         {
759             pm.pages[pmCounter]=disk.diskPages[diskIndex]; //get page from disk
760             vm.pagetable[vmIndex].physicalFrame=pmCounter;
761             vm.pagetable[vmIndex].rBit=1;
762             vm.pagetable[vmIndex].validBit=1;
763             ffolist[pmCounter]=disk.diskPages[diskIndex];
764             pmCounter++;
765         }
766     }
767 }

```

```

771         firstInFirstOut(arr[j]);
772     }
773     else if(algorithm==2){
774         secondC(arr[j]);
775     }
776     else{
777         LRUsoftware(arr[j]);
778     }
779 }
780 }
781
782 //-----
783 arr[j + 1] = arr[j];
784 j = j - 1;
785 //-----
786 }
787
788 if (!findInMemory(arr[j])) //Page replacement for arr[j] + Eğer memory dolu değilse direkt memory'e ekleme yapılır.
789 {
790     int vmIndex=findInVM(arr[j]);
791     int diskIndex=findInDisk(arr[j]);
792     if (pmCounter<memorySize)
793     {
794         pm.pages[pmCounter]=disk.diskPages[diskIndex]; //get page from disk
795         vm.pagetable[vmIndex].physicalFrame=pmCounter;
796         vm.pagetable[vmIndex].rBit=1;
797         vm.pagetable[vmIndex].validBit=1;
798         fifoList[pmCounter]=disk.diskPages[diskIndex];
799         pmCounter++;
800     }
801     else{
802         if (algorithm==1) //Eğer memory doluysa algoritmaya göre page replacement yapılır.
803         {
804             firstInFirstOut(arr[j]);
805         }
806         else if(algorithm==2){
807             secondC(arr[j]);
808         }
809         else{
810             LRUsoftware(arr[j]);
811         }
812     }
813 }
814 arr[j + 1] = key;
815 }
816 }
817

```

Classic insertion sort is applied. The replacement rules that I applied in the previous algorithm were also applied.

Bubble Sort

```

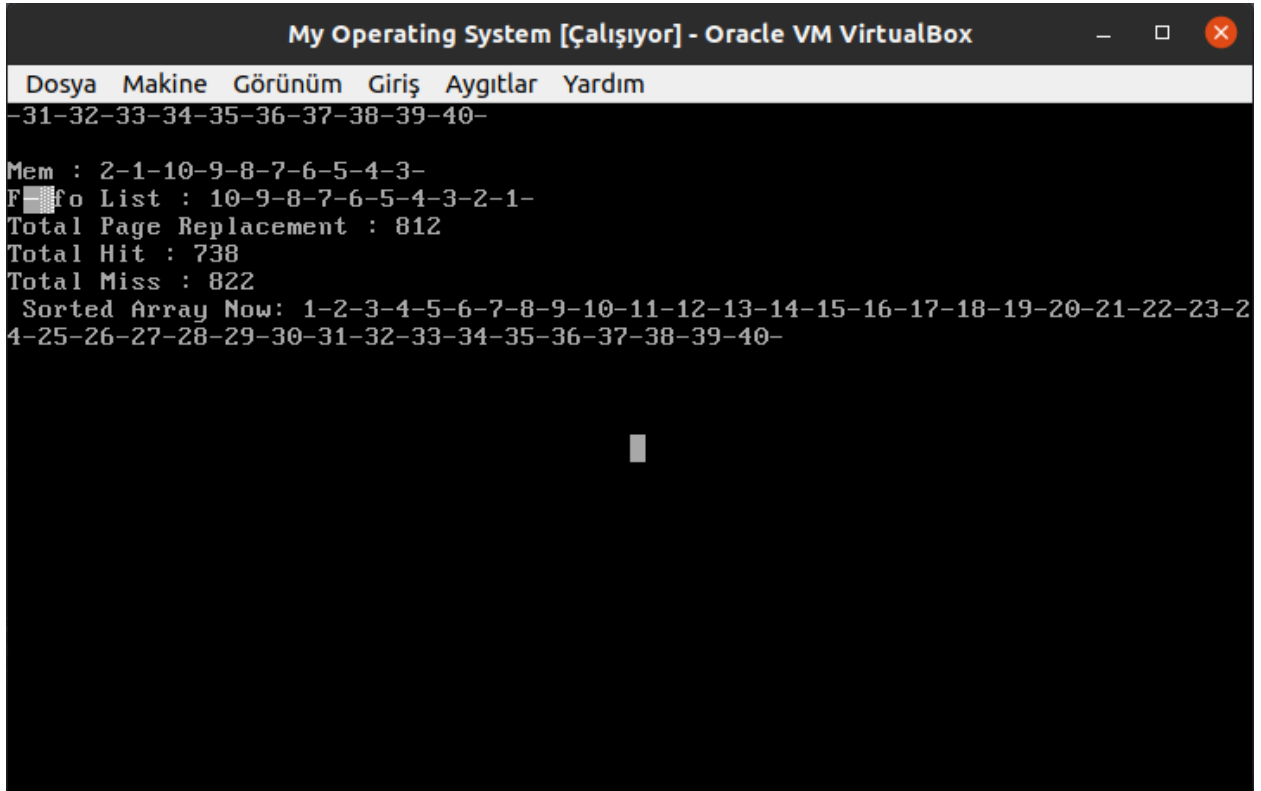
473 //-----SORT ALGORITHM-----
474 void bubbleSort(int arr[], int n,int algorithm){           // bubble sort function
475
476     for(int i = 0; i < n; i++){
477         for(int j = 0; j < n-i-1; j++){                    // loop until n-i-1(further is unnecessary)
478
479             if (!findInMemory(arr[j])) //Page replacement for arr[j] + Eğer memory dolu değilse direkt memory'e ekleme yapılır.
480             {
481                 int vmIndex=findInVM(arr[j]);
482                 int diskIndex=findInDisk(arr[j]);
483                 if (pmCounter<memorySize)
484                 {
485                     pm.pages[pmCounter]=disk.diskPages[diskIndex]; //get page from disk
486                     vm.pagetable[vmIndex].physicalFrame=pmCounter;
487                     vm.pagetable[vmIndex].rBit=1;
488                     vm.pagetable[vmIndex].validBit=1;
489                     fifoList[pmCounter]=disk.diskPages[diskIndex];
490                     pmCounter++;
491                 }
492             }
493             else{                                           //Eğer memory doluysa algoritmaya göre page replacement yapılır.
494                 if (algorithm==1)
495                 {
496                     firstInFirstOut(arr[j]);
497                 }
498                 else if(algorithm==2){
499                     secondC(arr[j]);
500                 }
501                 else{
502                     LRUSoftware(arr[j]);
503                 }
504             }
505         }
506     } if (!findInMemory(arr[j+1])) //Page Replacement for arr[j+1] + Eğer memory dolu değilse direkt memory'e ekleme yapılır.
507     {
508         int vmIndex=findInVM(arr[j+1]);
509         int diskIndex=findInDisk(arr[j+1]);
510         if (pmCounter<memorySize)
511         {
512             pm.pages[pmCounter]=disk.diskPages[diskIndex]; //get page from disk
513             vm.pagetable[vmIndex].physicalFrame=pmCounter;
514             vm.pagetable[vmIndex].rBit=1;
515             vm.pagetable[vmIndex].validBit=1;
516             fifoList[pmCounter]=disk.diskPages[diskIndex];
517             pmCounter++;
518         }
519         else{
520             //Eğer memory doluysa algoritmaya göre page replacement yapılır.
521             if (algorithm==1)
522             {
523                 firstInFirstOut(arr[j+1]);
524             }
525             else if(algorithm==2){
526                 secondC(arr[j+1]);
527             }
528             else{
529                 LRUSoftware(arr[j+1]);
530             }
531         }
532     }
533
534     if( arr[j] > arr[j+1]){
535         int temp = arr[j];
536         arr[j] = arr[j+1];
537         arr[j+1] = temp;
538     }
539 }
540 }
541 }
542 }

```

Classic bubble sort is applied. The replacement rules that I applied in the first algorithm were also applied.

SCREEN S.

FİFO - BUBBLE



```

My Operating System [Çalışıyor] - Oracle VM VirtualBox
Dosya Makine Görünüm Giriş Aygıtlar Yardım
-31-32-33-34-35-36-37-38-39-40-
Mem : 2-1-10-9-8-7-6-5-4-3-
Fifo List : 10-9-8-7-6-5-4-3-2-1-
Total Page Replacement : 812
Total Hit : 738
Total Miss : 822
Sorted Array Now: 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-

```

There are sorted versions of Array, some information, and final versions of fifo list memory.

Second Change - BUBBLE

```

My Operating System [Duraklatıldı] - Oracle VM VirtualBox
Dosya Makine Görünüm Giriş Aygıtlar Yardım
Fif: 1-11-10-9-8-7-6-5-4-3-
Mem: 1-11-10-9-8-7-6-5-4-3-
Fif: 1-11-10-9-8-7-6-5-4-3-
Fif: 11-10-9-8-7-6-5-4-3-2-
Mem: 2-11-10-9-8-7-6-5-4-3-
Fif: 11-10-9-8-7-6-5-4-3-2-
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-
Mem : 2-1-10-9-8-7-6-5-4-3-
Fifo List : 10-9-8-7-6-5-4-3-2-1-
Total Page Replacement : 812
Total Hit : 738
Total Miss : 822
Sorted Array Now: 1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-16-17-18-19-20-21-22-23-2
4-25-26-27-28-29-30-31-32-33-34-35-36-37-38-39-40-

```

There are sorted versions of Array, some information, and final versions of fifo list memory.

LRU - BUBBLE

```

My Operating System [Duraklatıldı] - Oracle VM VirtualBox
Dosya Makine Görünüm Giriş Aygıtlar Yardım
Mem: 40-39-38-37-36-35-34-33-32-2-
Fif: 40-39-38-37-36-35-34-33-32-31-
Mem: 40-39-38-37-36-35-34-33-32-4-
Fif: 40-39-38-37-36-35-34-33-32-31-
Mem: 40-39-38-37-36-35-34-33-32-1-
Fif: 40-39-38-37-36-35-34-33-32-31-
Mem: 40-39-38-37-36-35-34-33-32-3-
Fif: 40-39-38-37-36-35-34-33-32-31-
Mem: 40-39-38-37-36-35-34-33-32-2-
Fif: 40-39-38-37-36-35-34-33-32-31-
Mem: 40-39-38-37-36-35-34-33-32-3-
Fif: 40-39-38-37-36-35-34-33-32-31-
Mem: 40-39-38-37-36-35-34-33-32-1-
Fif: 40-39-38-37-36-35-34-33-32-31-
Mem: 40-39-38-37-36-35-34-33-32-2-
Fif: 40-39-38-37-36-35-34-33-32-31-
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-
Mem : 40-39-38-37-36-35-34-33-32-1-
Fifo List : 40-39-38-37-36-35-34-33-32-31-
Total Page Replacement : 1208
Total Hit : 342
Total Miss : 1218

```

There are sorted versions of Array, some information, and final versions of fifo list memory.

Second Chance - BUBBLE

```

My Operating System [Duraklatıldı] - Oracle VM VirtualBox
Dosya Makine Görünüm Giriş Aygıtlar Yardım
Fif: 1-11-10-9-8-7-6-5-4-3-
Mem: 1-11-10-9-8-7-6-5-4-3-
Fif: 1-11-10-9-8-7-6-5-4-3-
Fif: 11-10-9-8-7-6-5-4-3-2-
Mem: 2-11-10-9-8-7-6-5-4-3-
Fif: 11-10-9-8-7-6-5-4-3-2-
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-
Mem : 2-1-10-9-8-7-6-5-4-3-
Fifo List : 10-9-8-7-6-5-4-3-2-1-
Total Page Replacement : 812
Total Hit : 738
Total Miss : 822
Sorted Array Now: 1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-16-17-18-19-20-21-22-23-2-
4-25-26-27-28-29-30-31-32-33-34-35-36-37-38-39-40-

```

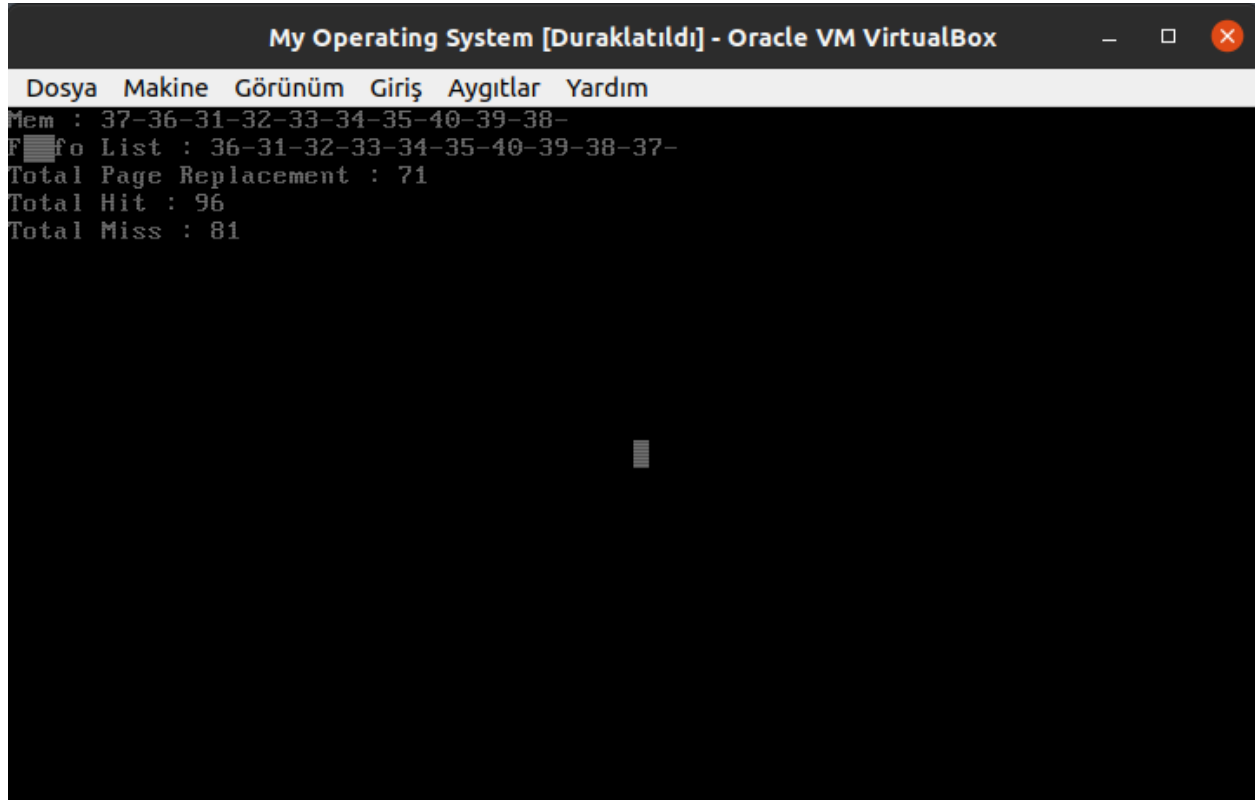
Second Chance - INSERTION

```

My Operating System [Duraklatıldı] - Oracle VM VirtualBox
Dosya Makine Görünüm Giriş Aygıtlar Yardım
Fif: 12-9-11-8-10-7-6-5-4-3-
Fif: 9-11-8-10-7-6-5-4-3-2-
Fif: 11-8-10-7-6-5-4-3-2-9-
Mem: 4-3-2-9-11-8-10-7-6-5-
Fif: 11-8-10-7-6-5-4-3-2-9-
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-
Mem : 4-3-2-9-0-8-10-7-6-5-
Fifo List : 8-10-7-6-5-4-3-2-9-0-
Total Page Replacement : 795
Total Hit : 872
Total Miss : 805

```

Second Chance - QUICK



```
My Operating System [Duraklatıldı] - Oracle VM VirtualBox
Dosya Makine Görünüm Giriş Aygıtlar Yardım
Mem : 37-36-31-32-33-34-35-40-39-38-
Ffo List : 36-31-32-33-34-35-40-39-38-37-
Total Page Replacement : 71
Total Hit : 96
Total Miss : 81
```

LRU - BUBBLE

```

My Operating System [Duraklatıldı] - Oracle VM VirtualBox
Dosya Makine Görünüm Giriş Aygıtlar Yardım
Mem: 40-39-38-37-36-35-34-33-32-2-
Fif: 40-39-38-37-36-35-34-33-32-31-
Mem: 40-39-38-37-36-35-34-33-32-4-
Fif: 40-39-38-37-36-35-34-33-32-31-
Mem: 40-39-38-37-36-35-34-33-32-1-
Fif: 40-39-38-37-36-35-34-33-32-31-
Mem: 40-39-38-37-36-35-34-33-32-3-
Fif: 40-39-38-37-36-35-34-33-32-31-
Mem: 40-39-38-37-36-35-34-33-32-2-
Fif: 40-39-38-37-36-35-34-33-32-31-
Mem: 40-39-38-37-36-35-34-33-32-3-
Fif: 40-39-38-37-36-35-34-33-32-31-
Mem: 40-39-38-37-36-35-34-33-32-1-
Fif: 40-39-38-37-36-35-34-33-32-31-
Mem: 40-39-38-37-36-35-34-33-32-2-
Fif: 40-39-38-37-36-35-34-33-32-31-
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-
Mem : 40-39-38-37-36-35-34-33-32-1-
Fifo List : 40-39-38-37-36-35-34-33-32-31-
Total Page Replacement : 1208
Total Hit : 342
Total Miss : 1218

```

LRU- INSERTION

```

My Operating System [Duraklatıldı] - Oracle VM VirtualBox
Dosya Makine Görünüm Giriş Aygıtlar Yardım
Mem: 39-40-38-37-36-35-34-33-3-0-
Fif: 39-40-38-37-36-35-34-33-32-31-
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-
Mem : 39-40-38-37-36-35-34-33-2-0-
Fifo List : 39-40-38-37-36-35-34-33-32-31-
Total Page Replacement : 526
Total Hit : 1141
Total Miss : 536

```

LRU - QUICK

```

My Operating System [Duraklatıldı] - Oracle VM VirtualBox
Dosya Makine Görünüm Giriş Aygıtlar Yardım
Fif: 20-11-1-2-3-4-5-6-7-8-
Mem: 33-30-40-32-34-35-36-37-38-39-
Fif: 20-11-1-2-3-4-5-6-7-8-
Mem: 33-29-40-32-34-35-36-37-38-39-
Fif: 20-11-1-2-3-4-5-6-7-8-
Mem: 33-27-40-32-34-35-36-37-38-39-
Fif: 20-11-1-2-3-4-5-6-7-8-
Mem: 33-26-40-32-34-35-36-37-38-39-
Fif: 20-11-1-2-3-4-5-6-7-8-
Mem: 33-29-40-32-34-35-36-37-38-39-
Fif: 20-11-1-2-3-4-5-6-7-8-
Mem: 33-28-40-32-34-35-36-37-38-39-
Fif: 20-11-1-2-3-4-5-6-7-8-
Mem: 33-30-40-32-34-35-36-37-38-39-
Fif: 20-11-1-2-3-4-5-6-7-8-
Mem: 33-29-40-32-34-35-36-37-38-39-
Fif: 20-11-1-2-3-4-5-6-7-8-
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-
Mem : 33-31-40-32-34-35-36-37-38-39-
Fifo List : 20-11-1-2-3-4-5-6-7-8-
Total Page Replacement : 96
Total Hit : 71
Total Miss : 106

```

General Summary

In this algorithm, I get an array to be sorted. Then I find each element of this array as a page and initialize it to virtual memory and disk. Then the sort algorithm and page replacement algorithm to be used are selected. Then, if each index used in the array is not in the memory, it is brought from the disk and the page table is updated. Here, the element in the index of the array is represented by page. If the used page is in memory, it is used directly. If not, and there is space in the memory, it is added to the memory and used. If the used page is in memory, it is used directly. If not, and there is space in the memory, it is added to the memory and used. This continues until the entire array has been sorted.