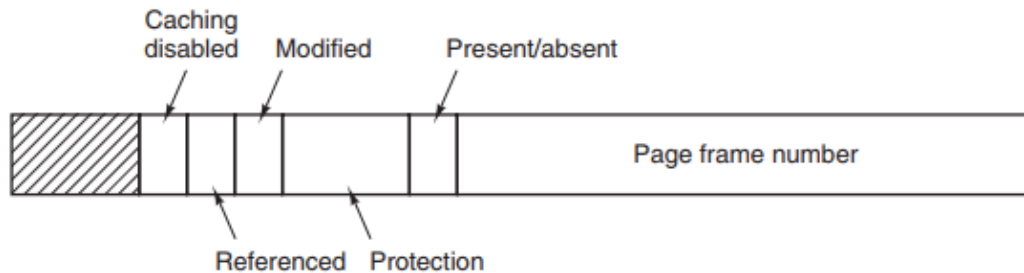


Operating Systems Spring 2020

Final Exam Project Report



Frame Number

It gives the frame number in which the current page you are looking for is present. The number of bits required depends on the number of frames. Frame bit is also known as address translation bit.

Number of bits for frame = $\text{Size of physical memory} / \text{frame size}$

Present/Absent bit

Present or absent bit says whether a particular page you are looking for is present or absent. In case if it is not present, that is called Page Fault. It is set to 0 if the corresponding page is not in memory. Used to control page fault by the operating system to support virtual memory. Sometimes this bit is also known as **valid/invalid** bits.

Protection bit

Protection bit says that what kind of protection you want on that page. So, these bit for the protection of the page frame (read, write etc).

Referenced bit

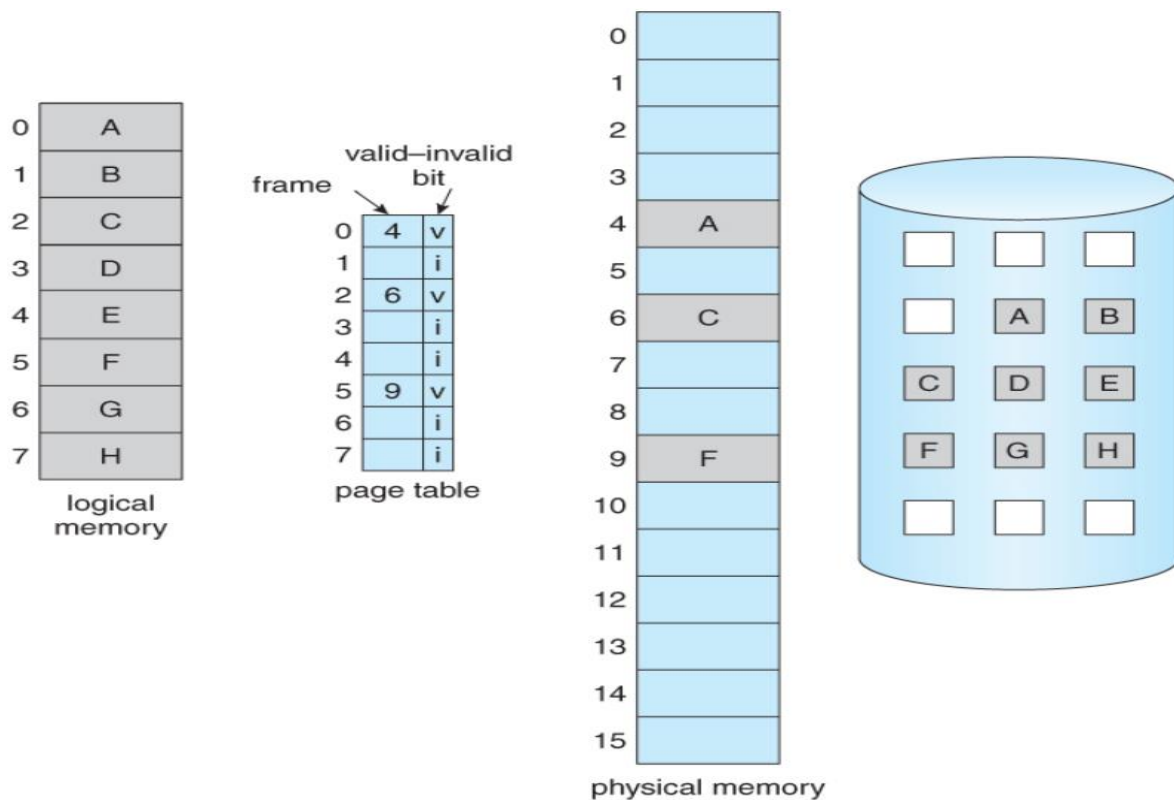
Referenced bit will say whether this page has been referred in the last clock cycle or not. It is set to 1 by hardware when the page is accessed

Modified bit

Modified bit says whether the page has been modified or not. Modified means sometimes you might try to write something on to the page. If a page is modified, then whenever you should replace that page with some other page, then the modified information should be kept on the hard disk or it has to be written back or it has to be saved back. It is set to 1 by hardware on write-access to page which is used to avoid writing when swapped out. Sometimes this modified bit is also called as the **Dirty bit**.

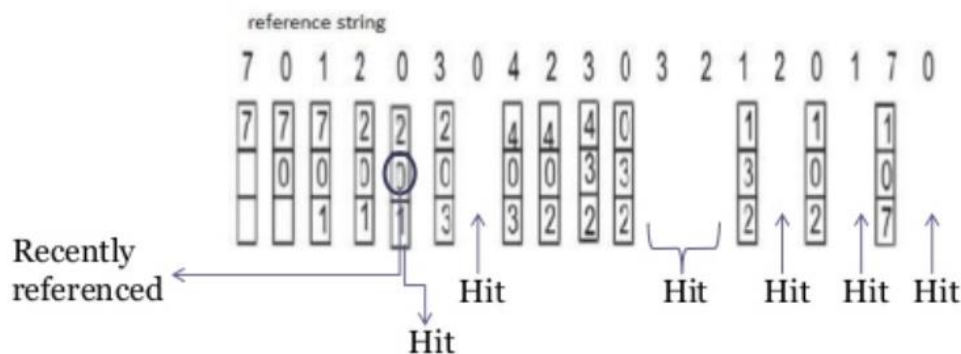
Caching enabled/disabled

Some times we need the fresh data. Let us say the user is typing some information from the keyboard and your program should run according to the input given by the user. In that case, the information will come into the main memory. Therefore main memory contains the latest information which is typed by the user. Now if you try to put that page in the cache, that cache will show the old information. So whenever freshness is required, we don't want to go for caching or many levels of the memory. The information present in the closest level to the CPU and the information present in the closest level to the user might be different. So we want the information has to be consistency, which means whatever information user has given, CPU should be able to see it as first as possible. That is the reason we want to disable caching. So, this bit **enables or disable** caching of the page



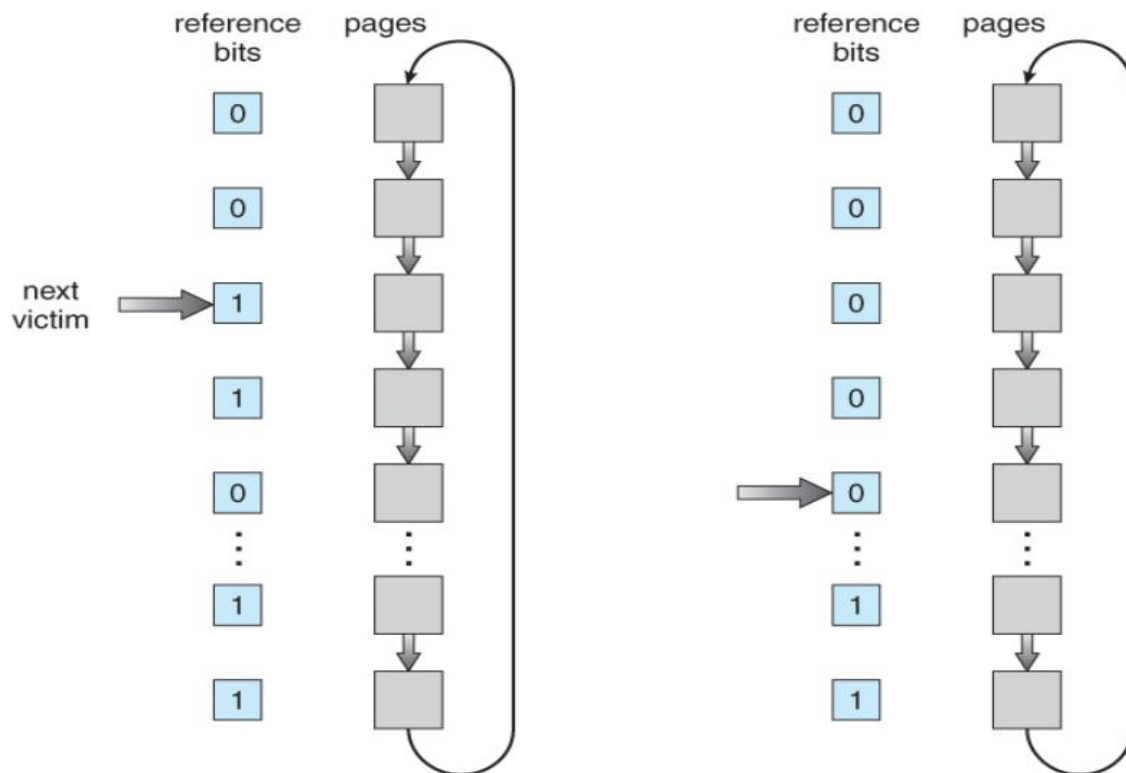
2)NRU

- It favours keeping pages in memory that have been recently used.
- The OS divides the pages into four classes based on usage during the last clock tick:
 - ◆ 3.Referenced,modified
 - ◆ 2.Referenced,not modified
 - ◆ 1.Not referenced,modified
 - ◆ 0.Not referenced,not modified
- Pick a random page from the lowest category for removal
- i.e. the not referenced,not modified page



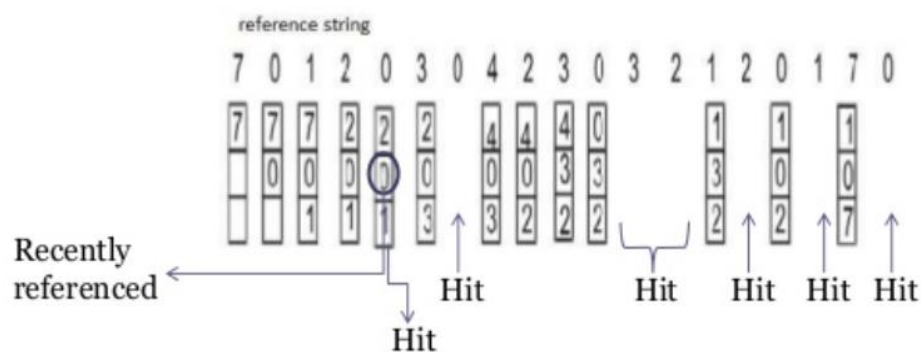
3)Second Chance

The second chance algorithm is essentially a FIFO, except the reference bit is used to give pages a second chance at staying in the page table. When a page must be replaced, the page table is scanned in a FIFO (circular queue) manner. If a page is found with its reference bit not set, then that page is selected as the next victim. If, however, the next page in the FIFO does have its reference bit set, then it is given a second chance: The reference bit is cleared, and the FIFO search continues. If some other page is found that did not have its reference bit set, then that page will be selected as the victim, and this page (the one being given the second chance) will be allowed to stay in the page table. If, however, there are no other pages that do not have their reference bit set, then this page will be selected as the victim when the FIFO search circles back around to this page on the second pass. If all reference bits in the table are set, then second chance degrades to FIFO, but also requires a complete search of the table for every page-replacement. As long as there are some pages whose reference bits are not set, then any page referenced frequently enough gets to stay in the page table indefinitely. This algorithm is also known as the clock algorithm, from the hands of the clock moving around the circular queue.



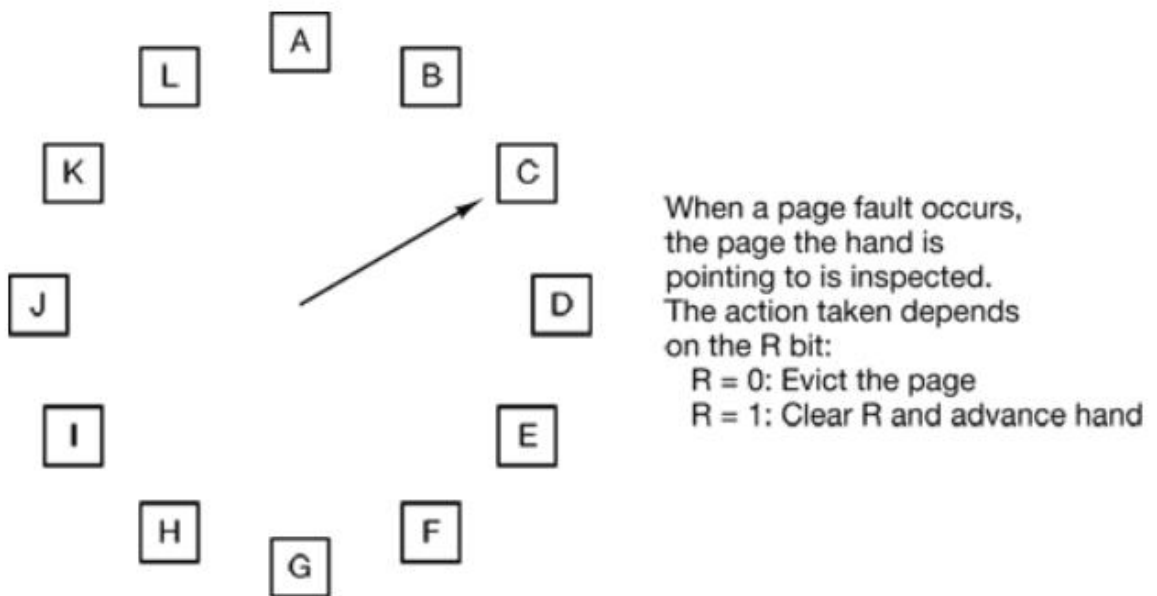
4)LRU

- It swaps the pages that have been used the least over a period of time
- It is free from Belady's anomaly



5) WSClock

Although second chance is a reasonable algorithm, it is unnecessarily inefficient because it is constantly moving pages around on its list. A better approach is to keep all the page frames on a circular list in the form of a clock. A hand points to the oldest page. When a page fault occurs, the page being pointed to by the hand is inspected. If its R bit is 0, the page is evicted, the new page is inserted into the clock in its place, and the hand is advanced one position. If R is 1, it is cleared and the hand is advanced to the next page. This process is repeated until a page is found with $R = 0$. Not surprisingly, this algorithm is called **clock**. It differs from second chance only in the implementation.



Implementation Some Code Part

Line numbers : 111-141

```
//fill physical memory and disk file
if(numVirtual >= numPhysical){
    int k=0;
    for(int i=0; i<numPhysical; ++i){
        for(int j=0; j<frameSize; ++j){
            int random = rand();
            physicalMemory[i][j] = random;
            address[k] = random;
            k++;
        }
        physicalSize++;
    }
    for(int i=frameSize*numPhysical; i<frameSize*numVirtual; ++i){
        int random = rand();
        fprintf(backing_store,"%d\n",random);
        address[k] = random;
        k++;
        fileSize++;
    }
}
else{
    int k=0;
    for(int i=0; i<numPhysical; ++i){
        for(int j=0; j<frameSize; ++j){
            int random = rand();
            physicalMemory[i][j] = random;
            address[k] = random;
            k++;
        }
    }
```

```
        physicalSize++;  
    } }
```

Line numbers : 143-151

```
//fill page table  
int k=0;  
for(int i=0; i<numVirtual; ++i){  
    for(int j=0; j<frameSize; ++j){  
        fill = i;  
        set(j,address[k],"fill");  
        k++;  
    }  
}
```

Line numbers 152-175

```
//create 4 sorting threads  
pthread_t Threads[4];  
int resize = 0;  
if(pthread_create(&Threads[0], NULL, bubbleSort, &resize) != 0){  
    perror("Pthread Create Error!\n");  
    exit(EXIT_FAILURE);  
}  
if(pthread_create(&Threads[1], NULL, mergeSort, &resize) != 0){  
    perror("Pthread Create Error!\n");  
    exit(EXIT_FAILURE);  
}  
if(pthread_create(&Threads[2], NULL, quickSort, &resize) != 0){  
    perror("Pthread Create Error!\n");  
    exit(EXIT_FAILURE);  
}
```



```

if(pthread_create(&Threads[3], NULL, indexSort, &resize) != 0){
    perror("Pthread Create Error!\n");
    exit(EXIT_FAILURE);
}
for(int j=0; j<4; ++j){
    if(pthread_join(Threads[j],NULL) != 0){
        perror("Pthread Join Error!\n");
        exit(EXIT_FAILURE);
    }
}

```

Line numbers : 177-186

```

//check if virtual memory is sorted
int wrongSort=0;
for(int i=0; i<numVirtual; ++i){
    check=i;
    for(int j=0; j<frameSize-1; ++j){
        for(int k=j+1; k<frameSize; ++k){
            if(get(j,"check") > get(k,"check"))
                wrongSort=1;
        }
    }
}

```

Line numbers 243-255

```

//prints page table at every pageTablePrintInt memory accesses
void pageTablePrint(){
    printf("Page Table:\n");
    for(int i=0;i<numVirtual;++i){
        printf("%d->",i);
        for(int j=0; j<frameSize;++j)
            printf("%d ",virtualMemory[i][j]);
    }
}

```

```

        printf("\n");
    }
    printf("\n");
    printf("\n");
}

```

Line numbers : 260-269

//looked value in physical memory.If value is available,wrote in virtual memory

```

for(int i=0; i<physicalSize; ++i){
    for(int j=0; j<frameSize; ++j){
        if(physicalMemory[i][j] == value){
            reads++;
            virtualMemory[fill][index] = value;
            writes++;
            found=1;
        }
    }
}

```

Line numbers : 279-331

//if value is not available in physical memory,looked in disk

//if value is available in disk,be page replacement

```

for(int i=0; i<fileSize; ++i){
    int read;
    fscanf(backing_store,"%d",&read);
    if(read == value){
        virtualMemory[fill][index] = value;
        diskRead++;
        char *tempName = "temp.txt";
        temp = fopen(tempName, "w");
        if (temp == NULL){
            fprintf(stderr, "Error opening %s\n",tempName);
            return;}
    }
}

```

```

fseek(backing_store, 0, SEEK_SET);
for(int j=0; j<fileSize; ++j){
    if(j == i){
        int write;
        fscanf(backing_store,"%d",&write);
        int result = pageReplacement(value);
        fprintf(temp,"%d\n",result);
        diskWrite++;
    }
    else{
        int write;
        fscanf(backing_store,"%d",&write);
        fprintf(temp,"%d\n",write);
    }
}
fclose(backing_store);
remove(diskFileName);
fclose(temp);
temp = fopen(tempName, "r");
if (temp == NULL){
    fprintf(stderr, "Error opening %s\n",tempName);
    return;
}
backing_store = fopen(diskFileName, "w");
if (backing_store == NULL){
    fprintf(stderr, "Error opening %s\n",diskFileName);
    return;
}

```

```

for(int j=0; j<fileSize; ++j){

    int write;

    fscanf(temp,"%d",&write);

    fprintf(backing_store,"%d\n",write);

}

fclose(temp);

remove(tempName);

return;

}

}

```

PART 2 OUTPUT

I run with ./sortArrays 3 3 4 LRU local 1000 diskFileName.dat

```

Page Table:
0->173226398 766020790 1071903604 1182770779 1333893513 1702255141 2121871803 2124051570
1->147856433 203709553 779257283 983886268 1364009855 1570801147 1653856994 1991873138
2->1220495216 1223105346 1538721691 1564659500 1634722517 1712204171 1742383952 1756404237
3->165749840 559409836 644531718 1133612654 1302352056 1405486716 1511256847 1899633444
4->43533414 488761921 643168494 666600572 1027419682 1742180616 2047247997 2072859842
5->487557985 770147550 973857104 1131370700 1263774204 1806676966 1954533399 2058359132
6->56251540 290991527 365095147 726271004 980629841 1257820733 1371709252 1999817665
7->578438982 935523245 1423570573 1567508387 1721615191 1821285695 2069135899 2126918223
8->1067200903 1746661889 240732116 1110734318 1646426238 883900610 2138154000 762716794
9->1371458595 1797347318 1532864344 1282334079 1604397070 359237800 266221131 828622674
10->724332948 992492135 1809252515 576666965 1283483662 1865504055 1834487698 71523259
11->1285528794 1110574623 2140659158 1264963370 1689013605 1814461205 839094913 608730861
12->912582036 1079827029 1187702587 1413639446 1675298830 1710135531 1719465179 1963727640
13->322553019 588774150 816912624 1060679527 1359999202 1419917327 1645535298 2144250275
14->1581266286 1307304165 573433592 717266300 1025324573 260437642 788789560 163369719
15->1371012266 781965070 1428333089 912542223 448942628 119944355 1521273084 1862582074

```

```

Page Table:
0->173226398 766020790 1071903604 1182770779 1333893513 1702255141 2121871803 2124051570
1->147856433 203709553 779257283 983886268 1364009855 1570801147 1653856994 1991873138
2->1220495216 1223105346 1538721691 1564659500 1634722517 1712204171 1742383952 1756404237
3->165749840 559409836 644531718 1133612654 1302352056 1405486716 1511256847 1899633444
4->43533414 488761921 643168494 666600572 1027419682 1742180616 2047247997 2072859842
5->487557985 770147550 973857104 1131370700 1263774204 1806676966 1954533399 2058359132
6->56251540 290991527 365095147 726271004 980629841 1257820733 1371709252 1999817665
7->578438982 935523245 1423570573 1567508387 1721615191 1821285695 2069135899 2126918223
8->240732116 762716794 883900610 1067200903 1110734318 1646426238 1746661889 2138154000
9->266221131 359237800 828622674 1282334079 1371458595 1532864344 1604397070 1797347318
10->71523259 576666965 724332948 992492135 1283483662 1809252515 1834487698 1865504055
11->608730861 839094913 1110574623 1264963370 1285528794 1689013605 1814461205 2140659158
12->912582036 1079827029 1187702587 1413639446 1675298830 1710135531 1719465179 1963727640
13->322553019 588774150 816912624 1060679527 1359999202 1419917327 1645535298 2144250275
14->163369719 260437642 573433592 717266300 788789560 1025324573 1307304165 1581266286
15->119944355 448942628 781965070 912542223 1371012266 1428333089 1521273084 1862582074

```

```

Sorting is successfull
For fill :
Number of reads : 64

```

Sorting is successfull
For fill :
Number of reads : 64
Number of writes : 64
Number of page misses : 64
Number of page replacements : 64
Number of disk page writes : 64
Number of disk page reads : 64

For bubble :
Number of reads : 64
Number of writes : 64
Number of page misses : 64
Number of page replacements : 64
Number of disk page writes : 64
Number of disk page reads : 64

For quick :
Number of reads : 64
Number of writes : 64
Number of page misses : 64
Number of page replacements : 64
Number of disk page writes : 64
Number of disk page reads : 64

For merge :
Number of reads : 64
Number of writes : 64
Number of page misses : 64
Number of page replacements : 64
Number of disk page writes : 64
Number of disk page reads : 64

For index :
Number of reads : 64
Number of writes : 64
Number of page misses : 64
Number of page replacements : 64
Number of disk page writes : 64

For index :
Number of reads : 64
Number of writes : 64
Number of page misses : 64
Number of page replacements : 64
Number of disk page writes : 64
Number of disk page reads : 64

For check :
Number of reads : 64
Number of writes : 64
Number of page misses : 64
Number of page replacements : 64
Number of disk page writes : 64
Number of disk page reads : 64

PART3

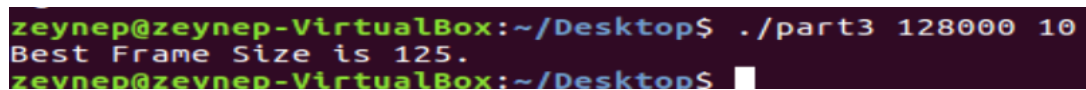
I tried all the possibilities to find the best frame size.

Code

```
//turn till found best frame size
while(numPhysical*frameSize < virtualMemCapacity){
    frameSize++;
    if(numPhysical*frameSize > virtualMemCapacity){
        frameSize--;
        break;
    }
}
printf("Best Frame Size is %d.\n",frameSize);
```

If the page fault rate exceeds a certain upper bound the that process needs more frames,and if it is below a given lower bound,then it can afford to give up some of its frames to other processes.

PART 3 OUTPUT

A terminal window with a dark background and green text. The prompt is 'zeynep@zeynep-VirtualBox:~/Desktop\$'. The command './part3 128000 10' has been entered. The output is 'Best Frame Size is 125.'. The prompt is now 'zeynep@zeynep-VirtualBox:~/Desktop\$' with a cursor.

```
zeynep@zeynep-VirtualBox:~/Desktop$ ./part3 128000 10
Best Frame Size is 125.
zeynep@zeynep-VirtualBox:~/Desktop$
```

GRAPHS

