KANAMPALLI HIMAJA
700732994
ASSIGNMENT - 2

1) Given six memory partitions of 300 KB, 600 KB, 350 KB, 200 KB, 750 KB, and 125 KB (in order), how would the first-fit, best-fit, and worst-fit algorithms place processes of size 115 KB, 500 KB, 358 KB, 200 KB, and 375 KB (in order)?

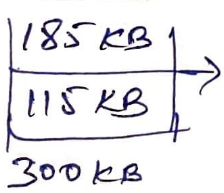
9.6)

Sol: First Fit: The First available Memory is allotted to the process.

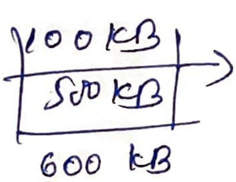
Best Fit: The memory in which least amount of size will be wasted or left over will be allotted.

Worst Case: The Largest Memory will be allotted. So, Applying the Algorithms.

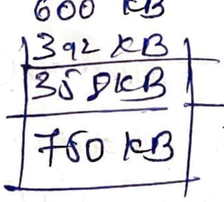
First Fit:



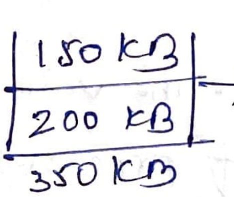
115 KB is put in 300 KB partition, leaving (185 KB, 600 KB, 350 KB, 200 KB, 750 KB, 125 KB)



500 KB is put in 600 KB partition, leaving (185 KB, 100 KB, 350 KB, 200 KB, 750 KB, 125 KB)



358 KB is put in 750 KB partition, leaving (185 KB, 100 KB, 350 KB, 200 KB, 392 KB, 125 KB)



200 KB is put in 350 KB partition, leaving (185 KB, 100 KB, 150 KB, 200 KB, 392 KB, 125 KB)

| 17 KB |
| 375 KB |
| 392 KB |

375 KB is put in 392 KB partition, leaving (185 KB, 100 KB, 150 KB, 200 KB, 17 KB, 125 KB)

Best Fit:

- 115 KB is put in 125 KB partition, leaving (300 KB, 600 KB, 350 KB, 200 KB, 750 KB, 10 KB)
- 500 KB is put in 600 KB partition, leaving (300 KB, 100 KB, 350 KB, 200 KB, 750 KB, 10 KB)
- 358 KB is put in 750 KB partition, leaving (300 KB, 100 KB, 350 KB, 200 KB, 392 KB, 10 KB)
- 200 KB is put in 200 KB partition, leaving (300 KB, 100 KB, 350 KB, 0 KB, 392 KB, 10 KB)
- 375 KB is put in 392 KB partition, leaving (300 KB, 100 KB, 350 KB, 0 KB, 17 KB, 10 KB)

Memory partitioning	300 KB	600 KB	350 KB	200 KB	750 KB	125 KB
Process sizes placed in		500 KB		200 KB	358 KB	115 KB
Having memory partitions	300 KB	100 KB	350 KB	0 KB	392 KB	10 KB

Worst Fit:

- 115 KB is put in 750 KB partition, leaving (300 KB, 600 KB, 350 KB, 200 KB, 635 KB, 125 KB)
- 500 KB is put in 635 KB partition, leaving (300 KB, 600 KB, 350 KB, 200 KB, 135 KB, 125 KB)
- 358 KB is put in 600 KB partition, leaving (300 KB, 242 KB, 350 KB, 200 KB, 135 KB, 125 KB)

- 200 KB is put in 350 KB partition, leaving (300 KB, 242 KB, 150 KB, 200 KB, 135 KB, 125 KB)
- 375 KB has to wait as no space is available having 375 KB of free Memory.

Memory partitions	300 KB	600 KB	350 KB	200 KB	750 KB	125 KB
process sizes		350 KB	200 KB		115 KB	
Placed in						
	300	342	150 KB	200 KB	635 KB	125 KB
					500	
					135 KB	

process of size 375 KB must wait.

⇒ Here we can say that Best Fit is the Best as it leaves the largest Holes after allotting the space but comparing the runtime of the processes.

Best Fit runs at time O(n) First Fit runs in Constant time O(1).

Q) Assuming a 1 KB page size, what are the page numbers and offsets for the following address references (provided as decimal numbers).

q.7

a) 3085 b) 42095 c) 215201 d) 650000 e) 2000001

Sol: Page number = address reference / Page size

offset = address reference % page size

a) (3085 / 1024) = 3, (3085 % 1024) = 13 (3, 13)

b) (42095 / 1024) = 41, (42095 % 1024) = 111 (41, 111).

c) (215201 / 1024) = 210, (215201 % 1024) = 161 (210, 161)

d) (650000 / 1024) = 634, (650000 % 1024) = 784 (634, 784)

e) (2000001 / 1024) = 512, (2000001 % 1024) = 1 (512, 1)

3) Under what circumstances do page faults occur? Describe the actions taken by the operating system when a page fault occurs.
10.1

Sol: **Page Fault:**

A page fault occurs when access to a page that has not been brought into the main memory takes place. [If the page fault occurs on the instruction fetch, we can restart by fetching the instruction again. If a page fault occurs while we are fetching an operand, we must fetch and decode the instruction again and then fetch the operand.

Actions by operating system:

⇒ When a page fault occurs, the operating system must bring the desired page from secondary storage into main memory.

→ Most operating system maintain a free-frame list, a pool of free frames for satisfying such requests (figure 1)

head → [7] → [97] → [15] → [126] ... → [75]

List of free frames

⇒ Operating systems typically allocate free frames using a technique known as "zero-fill-on-demand".

Steps to handle page fault:

1. Check an internal table for this process to determine whether the reference was a valid (or) an invalid memory access.

2. If the reference was invalid, we terminate the process. If it was valid but we have not yet brought in that page, we now page it in.

3. Find a frame

4. Schedule a secondary storage operation to read the desired page into the newly allocated frame.

5. When the storage read is complete, we modify the internal table kept with the process and the page table to indicate that the page is now in memory.

6. Restart the instruction that was interrupted by the trap. The process can now access the page as though it had always been in memory.

4) 10.2 Assume that you have a page-reference string for a process with m frames (initially all empty). The page reference string has length p, and n distinct page numbers occur in it. Answer these questions for any page-replacement algorithms.

a) what is a lower bound on the number of page faults?

Sol: Minimum could be "n" page faults because, each distinct page has to be in frame, if there is a repeating page number, that may not be added to frame, as it is already present.

b) What is an upper bound on the number of page faults?

sol: The maximum number of page faults would be "P" page faults.

Maximum is "P" because, it might so happen that every page number in reference string is not present in any of 'M' frames.