LAB 8 (Brownie Points)

Objective: Analyze belady's anomaly for different reference string lengths and different process sizes.

Belady's Anomaly: In general, if we increase the size of the frame, the number of page faults should decrease but we see it increasing in some cases. In this document, we will be seeing the belady's anomaly occurrence for reference string lengths from 25 to 12800 and process size varying from 20 to 100.

Code:

Code is attached in the zip folder. 2 ipynb files named FIFO Belady's anomaly and Belady's anomaly for random replacement techniques. 3 ipynb file contains the optimal, FIFO, LRU page replacement techniques.

Belady's anomaly for FIFO replacement technique

Code:

```
def BeladyAnomaly(reference_string, proc_size):

    page_faults = []

    for i in range(2,proc_size):
        frame_size = i
        queue = []
        temp_page_faults = 0
        for j in range(len(reference_string)):
            if reference_string[j] in queue:
                continue
```

```
        else:
            temp_page_faults += 1
            if(len(queue) < frame_size): //Checking if there is space in queue or not
                queue.append(reference_string[j])
            else:
                queue.pop(0)
                 queue.append(reference_string[i])
    page_faults.append(temp_page_faults)

  return page_faults
```

For this code, we send the reference string and the process size as the input and it returns the number of page faults.

- In this function, we are implementing FIFO concept
- If we find the number in reference string, we don't increase the page faults but if we don't find the number in the reference string, we increase the page faults variable by 1.
- If the queue is full, we dequeue or pop the first in element and append the latest number in the reference string.
- Page faults array returns different number of page faults varying frame size starting from 2 till process size sent to the function.

For various lengths and process sizes, we sent 1000 various reference strings and checked the number of bumps and appended it to a variable called bumps_arr.

And for the number of anomalous strings, i.e, number of strings which have bumps, it is stored in a variable anomaly_strings.

Refer to the FIFO Belady's random replacement ipynb file attached in the zip file for the code.

Note: The code takes a huge amount of time to run (>2hrs). The attached ipynb files have all the cells executed so you can directly refer to the outputs and graphs plotted in the notebooks.

Belady's anomaly for Random replacement technique

In this algorithm, we are removing elements from the list in a random manner, i.e, remove an element at random index. In this algorithm, we could see bumps and the code is attached below:

Code:

```
def BeladyAnomaly(reference_string, proc_size):

  page_faults = []

  for i in range(2,proc_size):
    frame_size = i
    queue = []
    temp_page_faults = 0
    for j in range(len(reference_string)):
      if reference_string[j] in queue:
        continue
      else:
        temp_page_faults += 1
        if(len(queue) < frame_size):
          queue.append(reference_string[j])
```

```
            else:
                index = random.randint(0,frame_size-1) //Random index for removal of element
                queue.pop(index)
                queue.append(reference_string[j])
        page_faults.append(temp_page_faults)

    return page_faults
```

So, in the above code, we can see the highlighted part where instead of popping the first element which we used in the FIFO case is not used here. Instead, with random integer generator, we are generating a random integer which is in between size of the queue and we are removing that element from the queue and appending the new page.

And for varying sizes of processes and reference string lengths, the code attached below was used. (Note: Only pasting the code used for FIFO replacement algorithm, the code is similar for Random replacement also).

Code:

```
anomaly_string = np.zeros((5,10), dtype = int)
bumps_arr = np.zeros((5,10), dtype = int)

rsl = 25
all_bumps_arr = []
anomalous_strings = []
m = 0
```

```
while(rsl<=12800):

  l = 0
  #-------------------------------------------------------------------------------
  for proc_size in range(20,120, 20):
    anomalous_stringcounts = 0
    total_bumps = 0
    #-------------------------------------------------------------------------------
    for j in range(1000): # taking 1000 random reference strings

      reference_string = generate_referenceString(rsl,proc_size)

      page_faults = BeladyAnomaly(reference_string, proc_size)

      bumps = 0
      for i in range(len(page_faults)-1):
        if page_faults[i] < page_faults[i+1]:
          bumps = bumps+1
      print(page_faults)
      print(bumps)


      if(bumps>0):
        anomalous_stringcounts += 1
        total_bumps += bumps
```

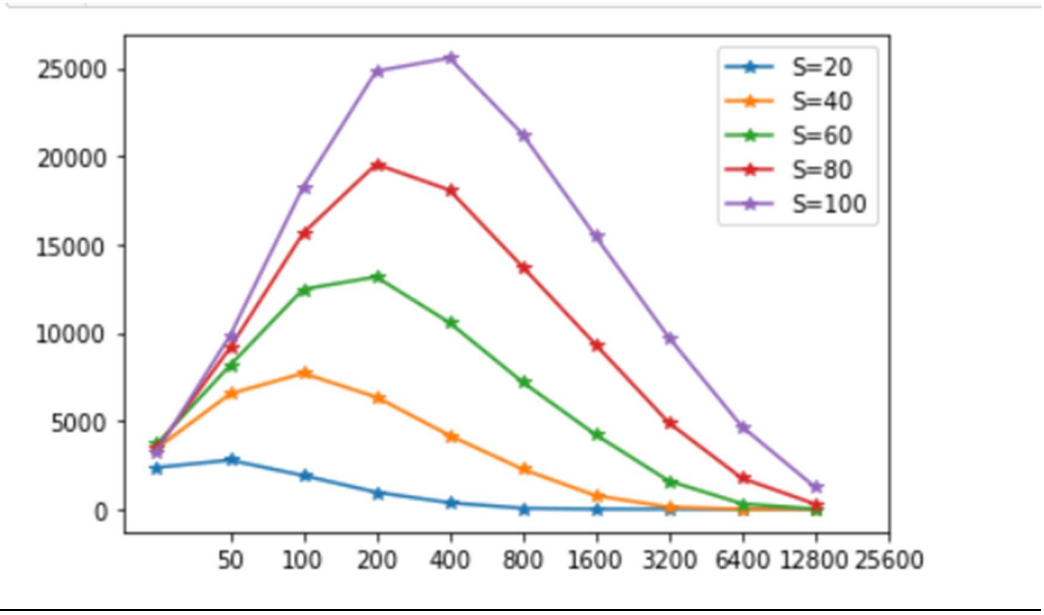
```
        #-------------------------------------------------------------------------------------------
        anomaly_string[l][m] = anomalous_stringcounts
        bumps_arr[l][m] = total_bumps
        l += 1


    m = m+1


    rsl = rsl * 2
```

For generating random reference strings, the below code was used.

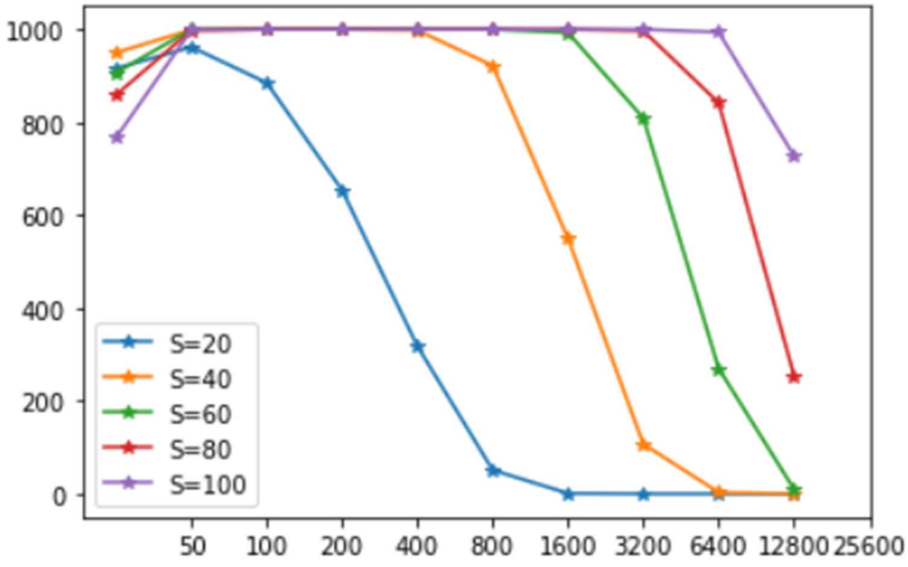
Code:

```
def generate_referenceString(length,process_size):
    string = []
    for i in range(length):
        n = random.randint(0,process_size-1)
        string.append(n)
    return string
```

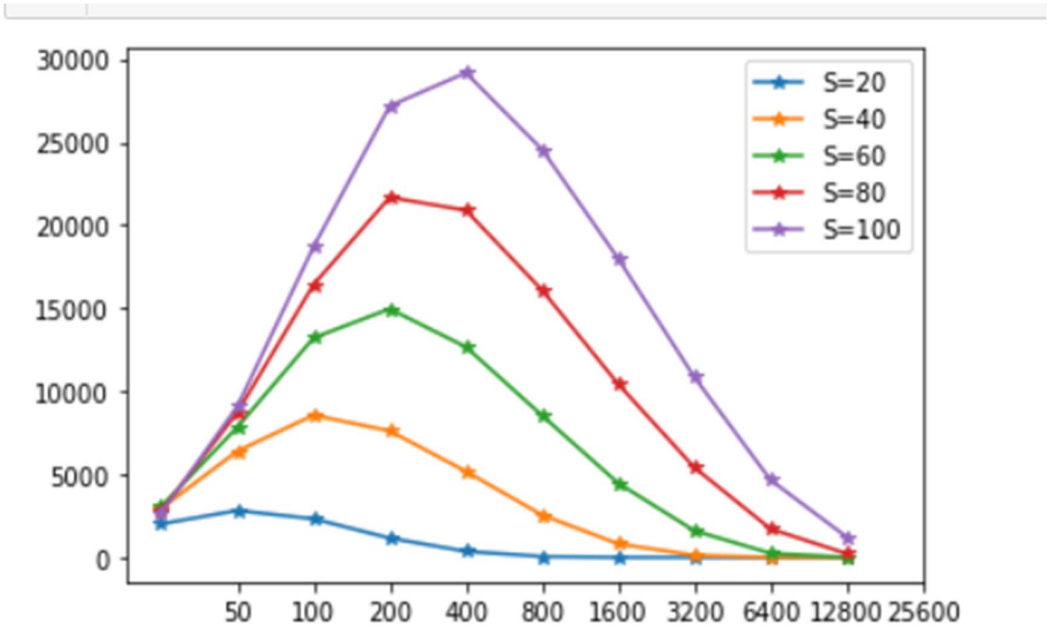
Graph for FIFO replacement technique and the bumps.



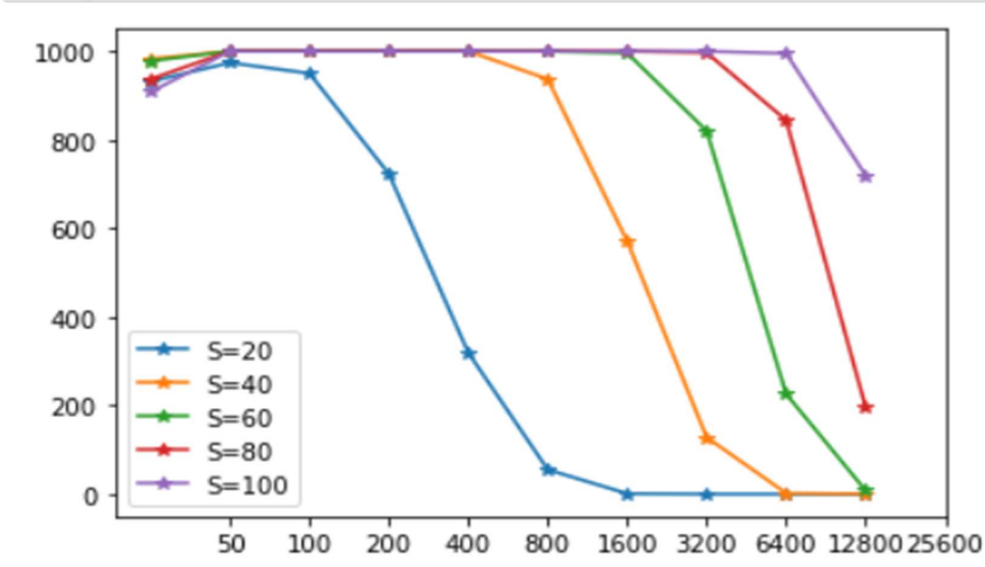
Graph for anomalous strings of FIFO replacement technique.



Graph for bumps and random replacement technique.



Graph for anomalous strings in random replacement technique.



Chaitanya Srikanth

SE20UARI038