Solution Details:

- Reading input file as per the input specification in the sample input file.

- As we read the input file, we create separate list (bank_details_may and bank_details_june) for month of May and June. We also create another list (bank_details_new) for new records.

- Display bank details (bank name and denomination count) of each list. The time complexity for this operation is **O(n).**

- Calculate the total count of Rs. 2000 for the months of May and June. The time complexity for this operation is **O(n).**

-

  a. Append new records to the lists – bank_details_may and bank_details_june according to the months. The time complexity for this operation is **O(n).**

  b. Create the max heap for the month of May and June using max_heap_build method. The time complexity for this operation is **O(nlog(n)).**

  Heapification:

```
Function max_heapify(bank_details, start_index, key_func){

        Left = 2*start_index + 1
        Right = 2 * start_index + 2
        Max_index = start_index

        If bank_deatils is not empty
                Size = len(bank_details)
                If Left < size and
                         key_func(bank_details[Left]) > key_func(bank_details[Max_index])
                        Max_index = Left
                If Right < size and
                         key_func(bank_details[Right]) > key_func(bank_details[Max_index])
                        Max_index = Right

                If Max_index != start_index then swap(bank_details, Max_index,
start_index)
                        If (Max_index == Right) then max_heapify
(bank_details,Right,key_func)
                        Else max_heapify (bank_details,Left,key_func)
        Else return null
}
```

  c. Maximum denomination count for each is the first element for that month (bank_details_may[0] and bank_details_june[0]). The time complexity for this operation is **O(1).**

- Sum the denomination count for all the banks from both the lists. The time complexity for this operation is **O(n).**

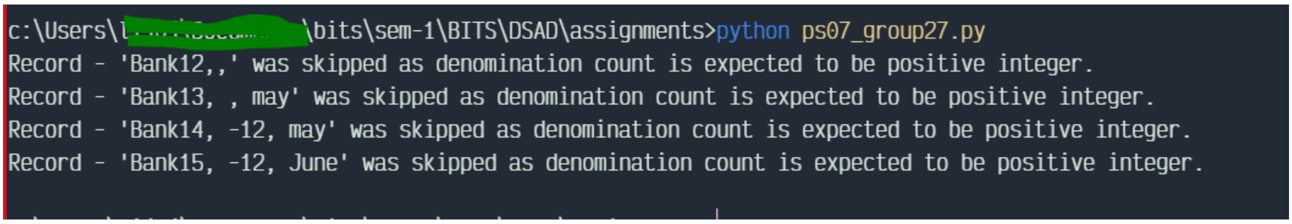
- Remove the first element from the may month maxheap (maxheap_pop function), and heapify the maxheap. This is done twice to remove the largest 2 elements. The time complexity for this operation is **O(log(n))**.

  Deletion:
  Function Max_heap_pop(bank_details, key_func){

  ```
  If length(bank_details) > 1
          Element = bank_details[0]
          bank_details[0] = bank_details.pop()
          max_heapify( bank_deatils,0,key_func)
  Else Element = bank_details.pop()
  Return Element;
  }
  ```

- Display bank details (bank name and denomination count) of each maxheap. The time complexity for this operation is **O(n).**

- *[Instruction 2]* Errors handling for invalid input has been implemented. These are logged to console. Underflow or empty list/heap is also handled in the code.



```
c:\Users\[redacted]\bits\sem-1\BITS\DSAD\assignments>python ps07_group27.py
Record - 'Bank12,,' was skipped as denomination count is expected to be positive integer.
Record - 'Bank13, , may' was skipped as denomination count is expected to be positive integer.
Record - 'Bank14, -12, may' was skipped as denomination count is expected to be positive integer.
Record - 'Bank15, -12, June' was skipped as denomination count is expected to be positive integer.
```

- *[Instruction 8.a]* We have used 2 separate maxheaps to store the bank details for each month. This ensure that the data is easy to access the details of largest denomination count (time complexity is **O(1)**). Also, when such element is removed, it is quite cheap to find the next largest denomination count and maintain the structure of the heap (time complexity is **O(nlog(n))**).

- *[Instruction 8.b]* Each operation and its time complexity.
  - ***build-maxheap:*** It iterates through non-leaf nodes starting the last one and call maxheapification. Choosing maxheap reduces the time complexity to **O(nlog(n))**, while with other structures, the sorting would be quite expensive with time complexity of $O(n^2)$.
  - ***maxheapification:*** maxheapification maintains the heap structure with largest element being at the top. This maxheapification process is of time complexity **O(nlog(n))**.
  - ***maxheap-pop:*** It removes the largest element from the heap and heapifies the remain structure. The time complexity of the process is **O(log(n))**.

- *[Instruction 8.c]* The ***alternate approach*** would be storing the bank details a ***stack*** based on the descending order of denomination count.
  - *Building* the sorted stack will be expensive with time complexity of $\mathbf{O(n^2)}$.
  - *Traversing* this stack will be time complexity **O(n)**.
  - *Popping* the largest element will be computationally cheap and have time complexity of **O(1)**.