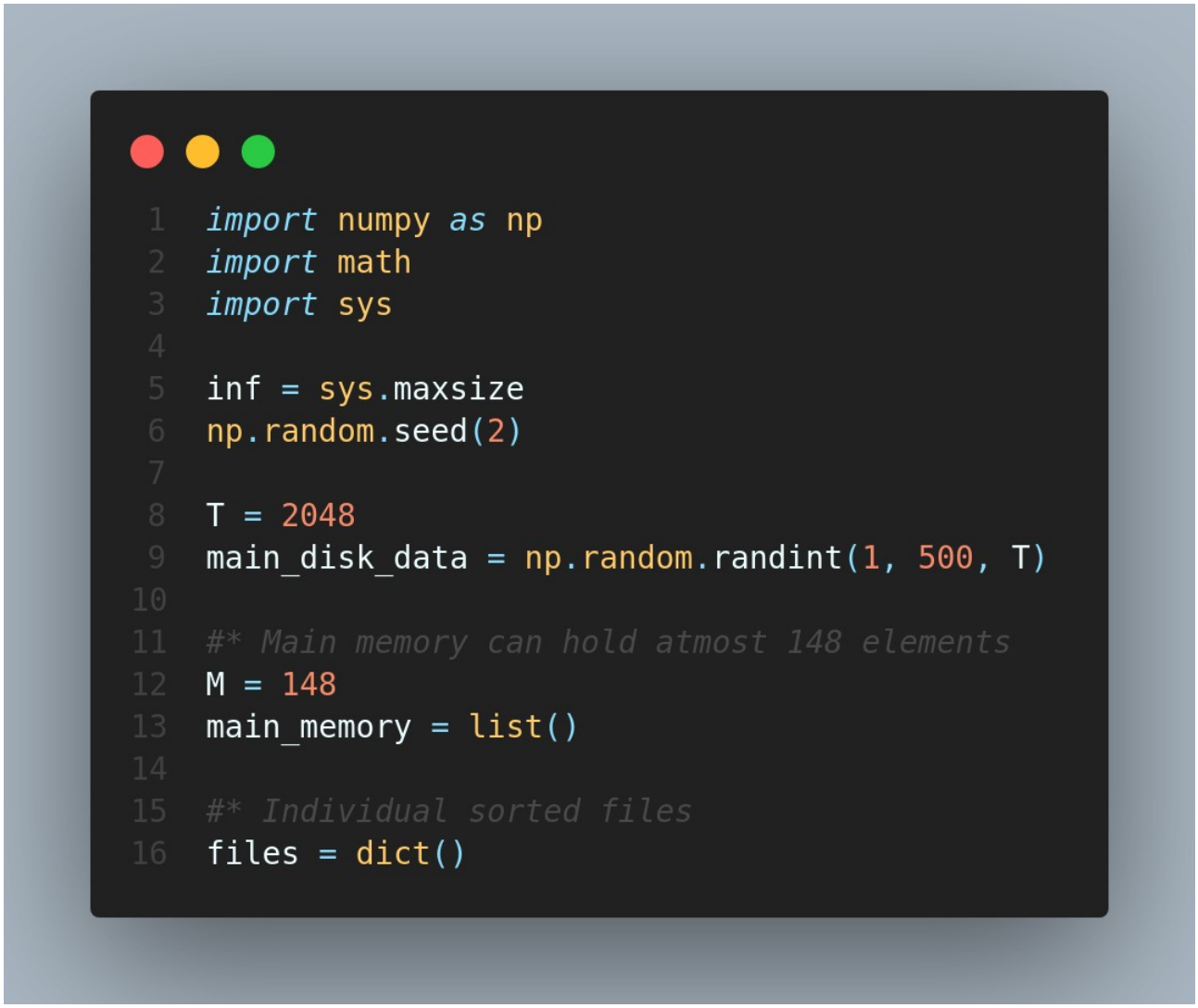
122022010 Shrayank Mistry
Description: Implementation of disk-based sorting using Quicksort and Min-Heap
Programming Language: Python

Random Numbers to sort 2048, Memory avaliable 148



```python
import numpy as np
import math
import sys

inf = sys.maxsize
np.random.seed(2)

T = 2048
main_disk_data = np.random.randint(1, 500, T)

#* Main memory can hold atmost 148 elements
M = 148
main_memory = list()

#* Individual sorted files
files = dict()
```

files data-structure is used to store sorted files.

Quicksort Algorithm to sort individual runs

```python
#TODO Phase I
#********************* QUICKSORT ALGORITHM ***********
***************#
def swap(A, i, j):
    temp = A[i]
    A[i] = A[j]
    A[j] = temp

def partition(A, L, H):
    pivot = A[L]
    i = L
    j = L + 1

    while j <= H:
        if A[j] <= pivot:
            i = i + 1
            swap(A, i, j)
        j = j + 1
    swap(A, L, i)
    return i

def quicksort(A, L, H):
    if L < H:
        pivotPos = partition(A, L, H)
        quicksort(A, L, pivotPos - 1)
        quicksort(A, pivotPos + 1, H)

def sorting(arr):
    N = len(arr)
    quicksort(arr, 0, N - 1)
    return arr
```

Making individual chunks and sorting them using quicksort.

```python
#********************** Creating Sorted Chunks ***********
************#
def make_sorted_files():
    file_counter = 1
    read_main_disk(files, file_counter)

def read_main_disk(files, file_counter):
    j = 0
    while j < len(main_disk_data):
        main_memory = main_disk_data[j:j + M]
        main_memory = sorting(main_memory)
        files[file_counter] = list(main_memory)
        file_counter += 1
        j = j + M
```

Data class is used to map every value to the run it belongs to.

```python
#TODO Phase II
#********************* Data Structures For Merging ***************
****#
#* Sorted Output on the disk
main_output = []

S = 0
sorted_output = []

class data:
    def __init__(self, value, id):
        self.value = value
        self.id = id

    def __str__(self):
        return '[' + str(self.value) + ', ' + str(self.id) + ']\t'
```

Min-Heap Data-structure to perform n-way merge of all the runs generated.

```
class Heap:
    def __init__(self):
        self.heap = [data(-1, 0)]

    def fillHeap(self, F, k):
        for i in range(F):
            for j in range(k):
                self.heap.append(data(files[i + 1][j], i + 1))
            index_dict[i + 1] += k

    def heapify(self, start_index, N):
        smallest = self.heap[start_index].value
        index = start_index
        L = 2 * start_index
        R = 2 * start_index + 1

        if (L <= N) and self.heap[L].value < smallest:
            index = L
            smallest = self.heap[L].value

        if (R <= N) and self.heap[R].value < smallest:
            index = R
            smallest = self.heap[R].value

        if (smallest != self.heap[start_index].value):
            temp = self.heap[start_index]
            self.heap[start_index] = self.heap[index]
            self.heap[index] = temp
            self.heapify(index, N)

    def merging(self, N):
        counter = 0
        while (self.heap[1].value != inf):
            if counter < S:
                sorted_output.append(self.heap[1].value)
                counter += 1
                id = self.heap[1].id

                if index_dict[id] >= len(files[id]): #* Elements in individual Array
                    self.heap[1].value = inf
                    self.heapify(1, N)
                else:
                    self.heap[1].value = files[id][index_dict[id]]
                    index_dict[id] += 1

                    self.heapify(1, N)
            else:
                for i in range(counter):
                    main_output.append(sorted_output[i])
                sorted_output.clear()
                counter = 0
        for i in range(counter):
            main_output.append(sorted_output[i])
        sorted_output.clear()
```

Main function for creating heap, filling the heap with intial run values and calling merge on them.

```python
#************************** Helper Functions and DS ********************
*#
#TODO Keep track on indexing of the files
index_dict = dict()
def create_index_dict(k):
    for i in range(k):
        index_dict[i + 1] = 0;

if __name__ == '__main__':
    make_sorted_files()
    F = math.ceil(T/M)

    H = len(files)
    S = M - H

    k = int(math.floor(H/F))
    heap_size = int(k * F)

    create_index_dict(F)
    h = Heap()

    h.fillHeap(F, k)

    #* Initial Heap Creation
    start_index = int(heap_size/2)

    while start_index > 0:
        h.heapify(start_index, heap_size)
        start_index -= 1

    h.merging(heap_size)
    print(main_output)

    print(len(main_output))
```

OUTPUT

```
RUNS OUTPUT (SIMULATION VALUES)

1 [8, 16, 23, 32, 35, 41, 44, 48, 50, 64, 73, 76, 76, 83, 86, 96]
2 [5, 21, 34, 38, 39, 40, 43, 47, 52, 59, 68, 68, 69, 70, 89, 91]
3 [5, 32, 40, 47, 50, 51, 53, 64, 67, 71, 77, 80, 81, 84, 91, 96]
4 [9, 9, 9, 16, 18, 23, 27, 33, 44, 51, 58, 63, 74, 84, 91, 97]
5 [2, 10, 11, 20, 35, 41, 41, 57, 61, 69, 71, 71, 77, 82, 83, 87]
6 [17, 19, 23, 44, 53, 62, 71, 73, 75, 85, 88, 91, 91, 92, 97, 98]
7 [16, 18, 22, 27, 31, 38, 40, 41, 44, 56, 60, 71, 75, 78, 81, 94]
8 [1, 21, 23, 28, 37, 41, 50, 51, 59, 60, 63, 64, 64, 74, 76, 93]
9 [2, 8, 33, 41, 47, 50, 50, 60, 74, 79, 80, 83, 92, 95, 96, 97]
10 [9, 12, 17, 22, 22, 44, 50, 52, 59, 64, 84, 87, 89, 93, 98, 99]
11 [2, 8, 18, 32, 35, 38, 43, 44, 58, 60, 64, 70, 80, 81, 81, 93]
12 [10, 18, 20, 22, 23, 37, 37, 37, 38, 40, 46, 49, 73, 84, 91, 96]
13 [9, 20, 28, 41, 50, 52, 57, 79]

SORTED OUTPUT

[1, 2, 2, 2, 5, 5, 8, 8, 8, 9, 9, 9, 9, 9, 10, 10, 11, 12, 16, 16, 16, 17, 17, 18, 18,
18, 18, 19, 20, 20, 20, 21, 21, 22, 22, 22, 22, 23, 23, 23, 23, 23, 27, 27, 28, 28, 31,
32, 32, 32, 33, 33, 34, 35, 35, 35, 37, 37, 37, 37, 38, 38, 38, 38, 39, 40, 40, 40, 40,
41, 41, 41, 41, 41, 41, 41, 43, 43, 44, 44, 44, 44, 44, 44, 46, 47, 47, 47, 48, 49, 50,
50, 50, 50, 50, 50, 50, 51, 51, 51, 52, 52, 52, 53, 53, 56, 57, 57, 58, 58, 59, 59, 59,
60, 60, 60, 60, 61, 62, 63, 63, 64, 64, 64, 64, 64, 64, 67, 68, 68, 69, 69, 70, 70, 71,
71, 71, 71, 71, 73, 73, 73, 74, 74, 74, 75, 75, 76, 76, 76, 77, 77, 78, 79, 79, 80, 80,
80, 81, 81, 81, 81, 82, 83, 83, 83, 84, 84, 84, 84, 85, 86, 87, 87, 88, 89, 89, 91, 91,
91, 91, 91, 91, 92, 92, 93, 93, 93, 94, 95, 96, 96, 96, 96, 97, 97, 97, 98, 98, 99]
```