Chp 3 - Exercise 2 - Individual

Use the code/pseudo code from Chp2 Exercise 1
•Draw a possible task dependency graph for merge sort where there are 16 items to be sorted.
    •For each task node specify what data is assigned to it.
    •How many total processors are needed?
    •Explain whether all the tasks are uniform in size?
    •What is the critical path for your graph and why?

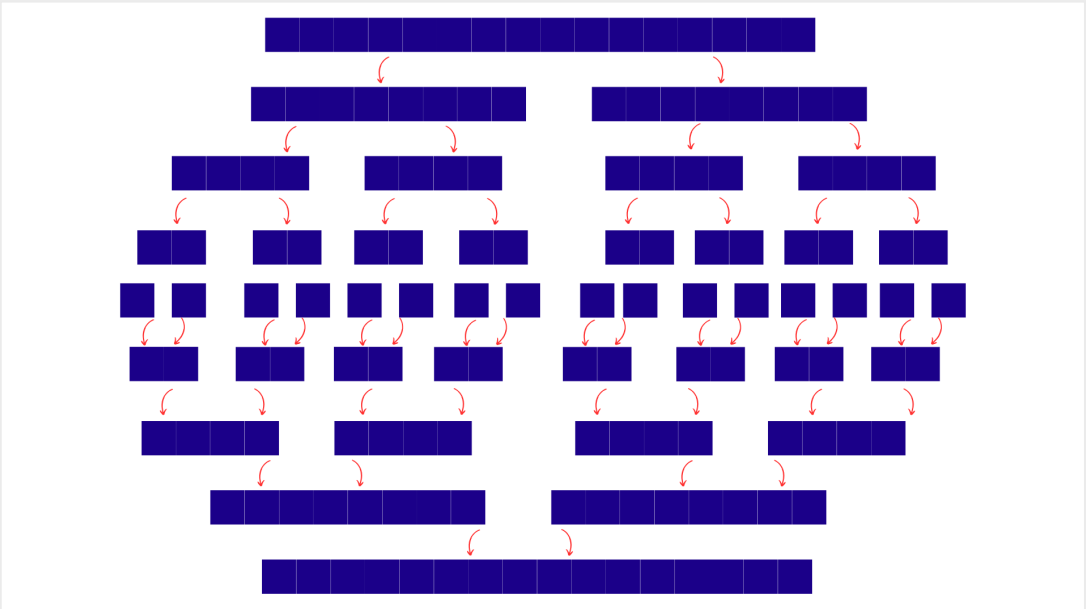
Grading Rubric:
_____ (1 Point) Provided code/pseudo code.
_____ (2 Points) Task nodes clearly state what data is assigned to them.
_____ (2 Points) Explain reasoning for # of processors needed.
_____ (1 Point) Reasoning for answer related to uniformity
_____ (2 Points) Critical Path length and reasoning

Answer:

```
# MergeSort in Python


def mergeSort(array):
    if len(array) > 1:

        #  r is the point where the array is divided into two
subarrays
        r = len(array)//2
        L = array[:r]
        M = array[r:]

        # Sort the two halves
        mergeSort(L)
        mergeSort(M)

        i = j = k = 0

        # Until we reach either end of either L or M, pick
larger among
        # elements L and M and place them in the correct
position at A[p..r]
```

```
        while i < len(L) and j < len(M):
            if L[i] < M[j]:
                array[k] = L[i]
                i += 1
            else:
                array[k] = M[j]
                j += 1
            k += 1

        # When we run out of elements in either L or M,
        # pick up the remaining elements and put in A[p..r]
        while i < len(L):
            array[k] = L[i]
            i += 1
            k += 1

        while j < len(M):
            array[k] = M[j]
            j += 1
            k += 1


# Print the array
def printList(array):
    for i in range(len(array)):
        print(array[i], end=" ")
    print()


# Driver program
if __name__ == '__main__':
    array = [6, 5, 12, 10, 9, 1]

    mergeSort(array)

    print("Sorted array is: ")
    printList(array)
```

Source: https://www.programiz.com/dsa/merge-sort



**Task Dependency Graph for 16 Items
** I didn't put numbers because its pain to graph on track pad compared to pen and paper

- Node 1 starts with all 16 items.
- This splits into two subnodes, each with 8 items.
- Each 8-item subnode splits further into two subnodes, each with 4 items.
- Each 4-item subnode splits again into two subnodes with 2 items.
- Each pair of 2-item subnodes is sorted and merged back into 4 items.
- The 4-item subnodes are then merged into 8-item subnodes.
- Finally, the two 8-item subnodes are merged into the full 16-item sorted array.

Number of Processors Needed

There are two options for parallelizing the process:
1. Using **4 processors**: One processor handles splitting the 16 items, then breaking and sorting 3 chunks while other processors sort their assigned parts simultaneously.
2. Using **8 processors**: Each processor handles a pair of items to break, sort, and merge. The maximum parallelization happens when sorting the 8 pairs before they are merged into larger subarrays.

Since the most parallel work occurs when sorting the 8 pairs (smallest subarrays), the most efficient option is **8 processors.**

Uniformity of Tasks

The tasks are **not uniform** in size. Early tasks involve splitting and merging larger

subarrays (e.g., 8 or 4 items), while later tasks are simpler, focusing on sorting and merging smaller subarrays (2 items). This uneven distribution of work is typical of the merge sort algorithm, where task complexity varies based on the subarray size being processed.

Critical Path

The critical path in the task dependency graph is the longest sequence of tasks that determines the total completion time. In merge sort, this corresponds to the sequence handling the largest subarrays, which requires the most operations.

The critical path follows this sequence:
- Splitting the 16-item array → Merging two 8-item arrays → Merging two 4-item arrays → Merging two 2-item arrays.

This path represents the deepest chain of dependent tasks. The length of the critical path is proportional to the number of levels of splitting and merging, which is **4 levels** (from 16 to 2 items, and back to 16). This is why it defines the total time to complete the merge sort operation.