Assignment 2

SFWRENG 2CO3: Data Structures and Algorithms–Winter 2023

Deadline: February 12, 2023

Department of Computing and Software
McMaster University

Please read the *Course Outline* for the general policies related to assignments.

Plagiarism is a *serious academic offense* and will be handled accordingly.
All suspicions will be reported to the *Office of Academic Integrity*
(in accordance with the Academic Integrity Policy).

This assignment is an *individual* assignment: do not submit work of others. All parts of your submission *must* be your own work and be based on your own ideas and conclusions. Only *discuss or share* any parts of your submissions with your TA or instructor. You are *responsible for protecting* your work: you are strongly advised to password-protect and lock your electronic devices (e.g., laptop) and to not share your logins with partners or friends! If you *submit* work, then you are certifying that you have completed the work for this assignment by yourself. By submitting work, you agree to automated and manual plagiarism checking of all submitted work.

Late submission policy. Late submissions will receive a late penalty of 20% on the score per day late (with a five hour grace period on the first day, e.g., to deal with technical issues) and submissions five days (or more) past the due date are not accepted. In case of technical issues while submitting, contact the instructor *before* the deadline.

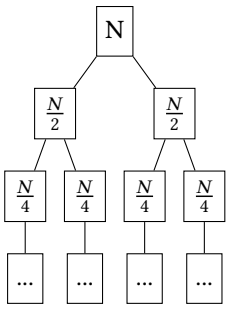
Problem 1. Assume we have a computer with infinite processor cores that can all operate at the same time. In an attempt to sort *as fast as possible*, we make a *parallelized* version of MergeSort that uses multiple processor cores at the same time. To achieve this, we change the top-down merge-sort algorithm as follows:

Algorithm ParallelMergeSort($L[start \ldots end]$)**:**

1: **if** $start + 1 < end$ **then**
2: $mid := (start + end)$ div 2.
3: Start ParallelMergeSort($L[start, mid]$) on a fresh processor core C.
4: L_2 := ParallelMergeSort($L[mid \ldots end]$).
5: Wait until C finished ParallelMergeSort($L[start, mid]$), resulting in L_1.
6: **return** Merge(L_1, L_2).
7: **else**
8: **return** L.
9: **end if**

Let $n = |L|$ be the length of the list being sorted by ParallelMergeSort.

P1.1. Give a recurrence $T(n)$ for the runtime complexity of ParallelMergeSort and solve the recurrence $T(n)$ by proving that $T(n) \sim e(n)$ for some expression e that uses n.
Answer:



Number	Cost	Total
1	N	$1 * N$
1	$\frac{N}{2}$	$1 * \frac{N}{2}$
1	$\frac{N}{4}$	$1 * \frac{N}{4}$
...	...	...
1	$\frac{N}{2^i}$	$1 * \frac{N}{2^i}$
—	—	—
		$2N$

As you can see, the tree shows that each step has $\frac{N}{2^i}$ cost. There is only 1 counted as they all occur in parallel, so unlike the cost complexity, the time complexity is as if the step is just run once. When you go to sum up all the steps, you can see that the steps will converge to 2N which is ~ N. Note: $N + \frac{N}{2} + \frac{N}{4} + \frac{N}{8} + ... = 2N$. Given N is not always a power of 2, we know that $O(N)$ is upper bounded by 2N and it must be lower bounded by N since the first step will always have a cost of N. This is enough to prove that the algorithm is $\sim N$.

P1.2. Provide a strict bound on the number of processor cores that ParallelMergeSort uses (hence, how many different processor cores are used by ParallelMergeSort at the same time).
Answer:
The number of processors is $2^{\log_2 N} = N$. This is because the recurrance tree proves to be $\log_2 N$ height and each step has double the processors of the last step. On the last reccurance level, it will terminate as there is only 1 element in the List, so it will hit the else statement (rather than the if condition) and return the List. This is an assumption that N is a power of 2. Given N is not always a power of 2, we know the number of processors is in the range of $2^{\log_2 N-1} <= processors <= N$, which contains the min and max number of processors needed per level on the reccurance tree.

Problem 2. Consider the following recursive sorting algorithm.

Algorithm WeirdSort(L, *start*, *end*)**:**

```
1:  if end − start = 2 then
2:    if L[start] ≥ L[start + 1] then
3:      Exchange L[start] and L[start + 1].
4:    end if
5:  else if end − start > 2 then
6:    k := (end − start) div 3.
7:    WeirdSort(L, start, end − k).
8:    WeirdSort(L, start + k, end).
9:    WeirdSort(L, start, end − k).
10: end if
```

Let $n = |L|$ be the length of the list being sorted by WeirdSort.

P2.1. Is WeirdSort a *stable* sort algorithm? If yes, explain why. If no, show why not and indicate whether the algorithm can be made stable.

P2.2. Prove via induction that WeirdSort will sort list L.

P2.3. Give a recurrence $T(n)$ for the runtime complexity of WeirdSort and solve the recurrence $T(n)$ by proving that $T(n) \sim e(n)$ for some expression e that uses n.

Problem 3. Consider the following PARTITION algorithm used by QUICKSORT (this version of PARTITION is based on the algorithm from the slides with the for-loop replaced by a while-loop).

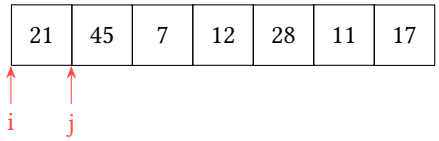
Algorithm PARTITION(L, *start*, *end*)**:**

1: $v, i, j := L[start], start, start + 1.$
2: **while** $j \neq end$ **do**
3: **if** $L[j] \leq v$ **then**
4: $i := i + 1.$
5: Exchange $L[i]$ and $L[j]$.
6: **end if**
7: $j := j + 1.$
8: **end while**
9: Exchange $L[i]$ and $L[start]$
10: **return** $i.$

P3.1. Illustrate the operations performed by PARTITION on the array $A = [21, 45, 7, 12, 28, 11, 17]$. Show the content of A after each execution of the loop body.

Answer:

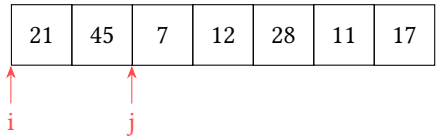
Start of function



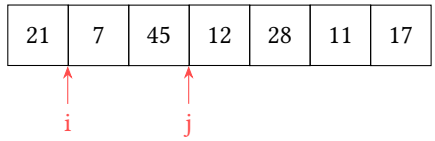
constant v = 21

Start of loop

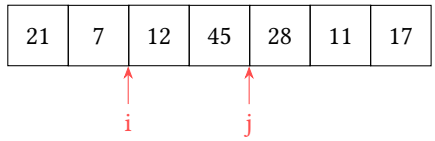
Loop 1



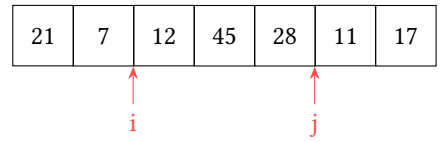
Loop 2



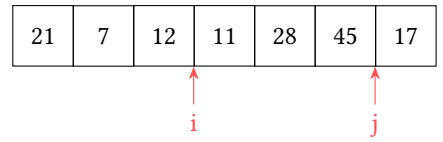
Loop 3



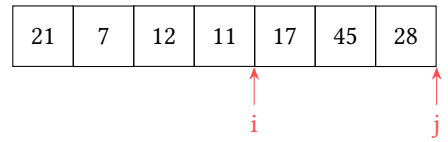
Loop 4



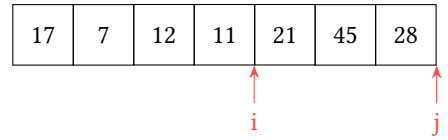
Loop 5



Loop 6



End of loop
Exchange L[i] and L[start]



P3.2. Provide pre-conditions and post-conditions for PARTITION and provide an invariant and bound function for the while-loop at Line 2. Prove the correctness of PARTITION.

P3.3. Argue how PARTITION can be adjusted to run on singly linked lists L, while keeping a running time of $\sim|L|$.

Problem 4. Consider pairs (x_i, y_i) such that x_i is the time at which person $i = 0, 1, 2, \ldots$ enters the museum and y_i is the time at which person i leaves the museum. You may assume that consecutive people enter the museum in order of increasing time ($x_0 \leq x_1 \leq \ldots$).

P4.1. Provide an algorithm MAXVISITORS that takes as input $L = [(x_0, y_0), \ldots, (x_{N-1}, y_{N-1})]$ and computes in $\sim N \log N$ the maximum number of visitors in the museum at any time.

P4.2. Argue why your algorithm MAXVISITORS is correct and has a runtime complexity of $\sim N \log N$.

P4.3. Assume that the museum has a maximum capacity of M. Provide a datastructure with an operation PERSONENTERS(x_i, y_i) that computes in at-most $\sim \log M$ the number of visitors in the museum when person i enters the museum (for any number of persons).

P4.4. Argue why your algorithm PERSONENTERS is correct and has a runtime complexity of $\sim \log M$.

Assignment Details

Write a report in which you solve each of the above problems. Your submission:

1. must be a PDF file;
2. must have clearly labeled solutions to each of the stated problems;

3. must be clearly presented;
4. must *not* be hand-written: prepare your report in LaTeX or in a word processor such as Microsoft Word (that can print or exported to PDF).

Submissions that do not follow the above requirements will get a grade of zero.

Grading

Each problem counts equally toward the final grade of this assignment.