**Summary**

**Group members:**

**20k-0477**

**20k-0385**

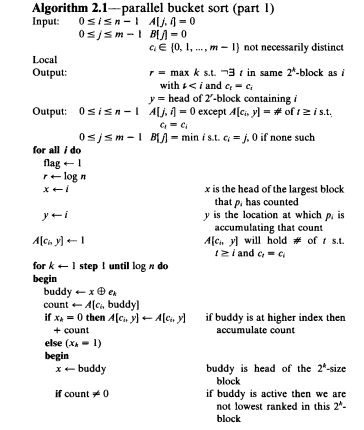
This algorithm presents bucket sort algorithm with time O(log n) and n processors is presented. The algorithm employs a technique that takes up more space than the sum of processors and time. A realistic model is used, with no memory contention allowed. There is often a time-space tradeoff in serial algorithms. In order to solve a problem within a certain time bound, a minimal amount of space is required. This space requirement may be reduced if we allow more time for the process. In the limit, there will be a minimum amount of data required.

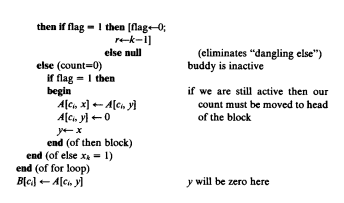
All processors in a computer have access to a shared memory as well as small local memories. All processors are synchronized and operate in accordance with the instructions of a single instruction stream. This model is known as a SIMD, and it serves as the foundation for our Computational Model of Programming. Our preliminary algorithm will sort n numbers using n (parallel) processors in time O(log n) under the assumption c n that the numbers that are to be sorted, { Ci}i=1, are from {0, 1 ..... m - 1} and with the provision that duplicate numbers are to be discarded. The provision will be dropped in the second algorithm.

There being n locations per area (numbered 0 through n - 1), each processor Pi can make its mark at location i in area Ci without fear of memory conflict. Iteratively, each processor then determines whether or not its "buddy" is active within the same area. (Here, "buddy" is defined analogously to the definition used in the Buddy System for dynamic memory allocation). It is noted that this bucket-sort algorithm requires space S = O(m n), time T = O(log n), and the use of n processors.

Another bucket sort algorithm will provide the actual ranking of the input numbers, with equal numbers remaining in the same relative order but receiving different ranks, assuming the input numbers come from a predefined small set.

It is noted that Algorithm 2 (the sequence of Algorithms 2.1, 2.2, 2.3) requires space S = O(m n), time T = O(log n + log m), and the use of n processors.





In a like manner, we can exhibit an algorithm to sort n numbers in O (klog n) time that uses n 1+1/k processors.

Interestingly, by setting k = log n (initially splitting the n elements into n/2 groups of 2 each), we obtain an algorithm to sort n numbers in O(log^2 n) time using O(n) processors, the same resources used by Batcher's algorithms.

Although these methods avoid memory-store conflicts, we observe that they do have memory-fetch conflicts. In other words, we provide simultaneous access to the same memory address by many processors.