Counting Inversions and Related Problems

By Timothy M Chan and Mihai Patrascu

Saran Neti

November 14, 2010

Outline

Outline

1. Introduction
   - Permutations
   - Inversions
2. Concepts
   - Offline/Online Algorithms
   - Radix Sort
   - Word RAM Model of computation
3. Results
   - History
   - The main result

What is a permutation?

- Given a set S, a permutation π of S is a set S' containing all elements of S, but in a different order.
- e.g. $\pi\{1,3,2\} = \{2,1,3\}$, $\pi\{1,3,2\} = \{1,2,3\}$ etc
- There are $n!$ permutations for a set of n elements.

What is a permutation?

- Given a set S, a permutation π of S is a set S' containing all elements of S, but in a different order.
- e.g. $\pi\{1,3,2\} = \{2,1,3\}$, $\pi\{1,3,2\} = \{1,2,3\}$ etc
- There are $n!$ permutations for a set of n elements.

What is a permutation?

- Given a set S, a permutation π of S is a set S' containing all elements of S, but in a different order.
- e.g. $\pi\{1,3,2\} = \{2,1,3\}$, $\pi\{1,3,2\} = \{1,2,3\}$ etc
- There are $n!$ permutations for a set of n elements.

What is a permutation?

- Given a set S, a permutation π of S is a set $S^{'}$containing all elements of S, but in a different order.
- e.g. $\pi\{1,3,2\} = \{2,1,3\}$, $\pi\{1,3,2\} = \{1,2,3\}$ etc
- There are $n!$ permutations for a set of n elements.

Outline

1. Introduction
   - Permutations
   - Inversions
2. Concepts
   - Offline/Online Algorithms
   - Radix Sort
   - Word RAM Model of computation
3. Results
   - History
   - The main result

What is an Inversion?

- The number of inversions in a permutation π is defined as the number of pairs $i < j$ with $\pi(i) > \pi(j)$
- e.g. The number of inversions in $\{1, 6, 2, 9, 5\} = 3$
  The actual sorted order is $\{1, 2, 5, 6, 9\}$
  The pair $\{6, 2\}, \{6, 5\}, \{9, 5\}$ are in the "wrong" order
- Inversion is a measure of deviation from a sorted order. We want to "flip" the inversion pairs to get the sorted order.
- Question - Given a permutation, how do you count the number of inversions in it ?
  i.e How messed up it is from a nice sorted order.

What is an Inversion?

- The number of inversions in a permutation π is defined as the number of pairs $i < j$ with $\pi(i) > \pi(j)$
- e.g. The number of inversions in $\{1,6,2,9,5\} = 3$
  The actual sorted order is $\{1,2,5,6,9\}$
  The pair $\{6,2\},\{6,5\},\{9,5\}$ are in the "wrong" order
- Inversion is a measure of deviation from a sorted order. We want to "flip" the inversion pairs to get the sorted order.
- Question - Given a permutation, how do you count the number of inversions in it ?
  i.e How messed up it is from a nice sorted order.

What is an Inversion?

- The number of inversions in a permutation π is defined as the number of pairs $i < j$ with $\pi(i) > \pi(j)$
- e.g. The number of inversions in $\{1,6,2,9,5\} = 3$
  The actual sorted order is $\{1,2,5,6,9\}$
  The pair $\{6,2\},\{6,5\},\{9,5\}$ are in the "wrong" order
- Inversion is a measure of deviation from a sorted order. We want to "flip" the inversion pairs to get the sorted order.
- Question - Given a permutation, how do you count the number of inversions in it ?
  i.e How messed up it is from a nice sorted order.

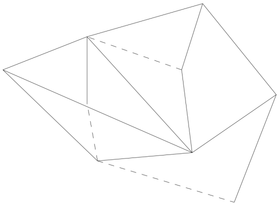
What is an Inversion?

- The number of inversions in a permutation π is defined as the number of pairs $i < j$ with $\pi(i) > \pi(j)$
- e.g. The number of inversions in $\{1,6,2,9,5\} = 3$
  The actual sorted order is $\{1,2,5,6,9\}$
  The pair $\{6,2\},\{6,5\},\{9,5\}$ are in the "wrong" order
- Inversion is a measure of deviation from a sorted order. We want to "flip" the inversion pairs to get the sorted order.
- Question - Given a permutation, how do you count the number of inversions in it ?
  i.e How messed up it is from a nice sorted order.

Outline

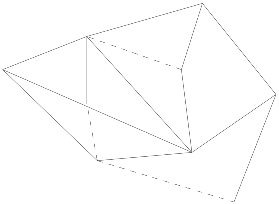
Offline/Online algorithms

- An online algorithm runs in a serial manner, and produces output as and when it receives input.
- An offline algorithm runs after the entire input has been received. Can Offline be better than Online?
- e.g Canadian Traveller's Problem - Given a graph with some unreliable (dotted) edges, find the shortest path to a destination. You'll know if an edge is unreliable when you reach vertex containing the edge.



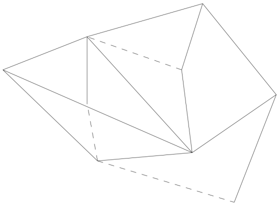
Offline/Online algorithms

- An online algorithm runs in a serial manner, and produces output as and when it receives input.
- An offline algorithm runs after the entire input has been received. Can Offline be better than Online?
- e.g Canadian Traveller's Problem - Given a graph with some unreliable (dotted) edges, find the shortest path to a destination. You'll know if an edge is unreliable when you reach vertex containing the edge.



Offline/Online algorithms

- An online algorithm runs in a serial manner, and produces output as and when it receives input.
- An offline algorithm runs after the entire input has been received. Can Offline be better than Online?
- e.g Canadian Traveller's Problem - Given a graph with some unreliable (dotted) edges, find the shortest path to a destination. You'll know if an edge is unreliable when you reach vertex containing the edge.



Outline

Radix Sort - Sorting without comparision

- Radix Sort sorts number based on the “radix” or “base”.
- e.g Sorting the following base-10 numbers: 170, 045, 075, 090, 002, 024, 802, 066
- Sort by Unit’s place - 170, 090, 002, 802, 024, 045, 075, 066
- Sort by 10s place - 002, 802, 024, 045, 066, 170, 075, 090
- Sort by 100s place - 002, 024, 045, 066, 075, 090, 170, 802
- For a set of n numbers, L bits each, Radix Sort takes O(nL) time.
- In case of binary representation, $L = log_2 n$, and we get the familiar $O(nlogn)$ time.

Radix Sort - Sorting without comparision

- Radix Sort sorts number based on the “radix” or “base”.
- e.g Sorting the following base-10 numbers: 170, 045, 075, 090, 002, 024, 802, 066
- Sort by Unit’s place - 170, 090, 002, 802, 024, 045, 075, 066
- Sort by 10s place - 002, 802, 024, 045, 066, 170, 075, 090
- Sort by 100s place - 002, 024, 045, 066, 075, 090, 170, 802
- For a set of n numbers, L bits each, Radix Sort takes O(nL) time.
- In case of binary representation, $L = log_2 n$, and we get the familiar $O(nlogn)$ time.

Radix Sort - Sorting without comparision

- Radix Sort sorts number based on the "radix" or "base".
- e.g Sorting the following base-10 numbers: 170, 045, 075, 090, 002, 024, 802, 066
- Sort by Unit's place - 170, 090, 002, 802, 024, 045, 075, 066
- Sort by 10s place - 002, 802, 024, 045, 066, 170, 075, 090
- Sort by 100s place - 002, 024, 045, 066, 075, 090, 170, 802
- For a set of n numbers, L bits each, Radix Sort takes O(nL) time.
- In case of binary representation, $L = log_2 n$, and we get the familiar $O(nlogn)$ time.

Radix Sort - Sorting without comparision

- Radix Sort sorts number based on the “radix” or “base”.
- e.g Sorting the following base-10 numbers: 170, 045, 075, 090, 002, 024, 802, 066
- Sort by Unit's place - 170, 090, 002, 802, 024, 045, 075, 066
- Sort by 10s place - 002, 802, 024, 045, 066, 170, 075, 090
- Sort by 100s place - 002, 024, 045, 066, 075, 090, 170, 802
- For a set of n numbers, L bits each, Radix Sort takes O(nL) time.
- In case of binary representation, $L = log_2 n$, and we get the familiar $O(nlogn)$ time.

Radix Sort - Sorting without comparision

- Radix Sort sorts number based on the “radix” or “base”.
- e.g Sorting the following base-10 numbers: 170, 045, 075, 090, 002, 024, 802, 066
- Sort by Unit’s place - 170, 090, 002, 802, 024, 045, 075, 066
- Sort by 10s place - 002, 802, 024, 045, 066, 170, 075, 090
- Sort by 100s place - 002, 024, 045, 066, 075, 090, 170, 802
- For a set of n numbers, L bits each, Radix Sort takes O(nL) time.
- In case of binary representation, $L = log_2 n$, and we get the familiar $O(nlogn)$ time.

Radix Sort - Sorting without comparision

- Radix Sort sorts number based on the “radix” or “base”.
- e.g Sorting the following base-10 numbers: 170, 045, 075, 090, 002, 024, 802, 066
- Sort by Unit's place - 170, 090, 002, 802, 024, 045, 075, 066
- Sort by 10s place - 002, 802, 024, 045, 066, 170, 075, 090
- Sort by 100s place - 002, 024, 045, 066, 075, 090, 170, 802
- For a set of n numbers, L bits each, Radix Sort takes O(nL) time.
- In case of binary representation, $L = log_2 n$, and we get the familiar $O(nlogn)$ time.

Radix Sort - Sorting without comparision

- Radix Sort sorts number based on the "radix" or "base".
- e.g Sorting the following base-10 numbers: 170, 045, 075, 090, 002, 024, 802, 066
- Sort by Unit's place - 170, 090, 002, 802, 024, 045, 075, 066
- Sort by 10s place - 002, 802, 024, 045, 066, 170, 075, 090
- Sort by 100s place - 002, 024, 045, 066, 075, 090, 170, 802
- For a set of n numbers, L bits each, Radix Sort takes O(nL) time.
- In case of binary representation, $L = log_2 n$, and we get the familiar $O(nlogn)$ time.

Outline

1. Introduction
   - Permutations
   - Inversions
2. Concepts
   - Offline/Online Algorithms
   - Radix Sort
   - Word RAM Model of computation
3. Results
   - History
   - The main result

Why Turing Machine?

- We don't use Turing Machines in practice...no tapes, symbols or transition functions, etc.
- Practical Computers use Hierarchical Memory organization. L1 Cache -> L2 Cache -> SRAM -> DRAM -> Hard Disk -> Tape Storage
- Faster memory is more expensive and vice versa.
- Can we build a more realistic computational model than a Turing machine?

Why Turing Machine?

- We don't use Turing Machines in practice...no tapes, symbols or transition functions, etc.
- Practical Computers use Hierarchical Memory organization. L1 Cache -> L2 Cache -> SRAM -> DRAM -> Hard Disk -> Tape Storage
- Faster memory is more expensive and vice versa.
- Can we build a more realistic computational model than a Turing machine?

Why Turing Machine?

- We don't use Turing Machines in practice...no tapes, symbols or transition functions, etc.
- Practical Computers use Hierarchical Memory organization. L1 Cache -> L2 Cache -> SRAM -> DRAM -> Hard Disk -> Tape Storage
- Faster memory is more expensive and vice versa.
- Can we build a more realistic computational model than a Turing machine?

Why Turing Machine?

- We don't use Turing Machines in practice...no tapes, symbols or transition functions, etc.
- Practical Computers use Hierarchical Memory organization. L1 Cache -> L2 Cache -> SRAM -> DRAM -> Hard Disk -> Tape Storage
- Faster memory is more expensive and vice versa.
- Can we build a more realistic computational model than a Turing machine?

Assumptions in a Word RAM model

- Memory is organized into words of size w.
  A word is 32 bits, or 64 bits in modern day computers.
- If n is the maximum size of the input to the algorithm, $w > log(n)$
- All normal (arithmetical/logical) computations are performed on a Word and they take O(1) time.
- Words can be accessed Randomly. (Random Access Memory).
- Computational Times for many problems can be improved in this model.
  e.g Sorting has been shown to take $O(n\sqrt{loglogn})$ time.

Assumptions in a Word RAM model

- Memory is organized into words of size w.
  A word is 32 bits, or 64 bits in modern day computers.
- If n is the maximum size of the input to the algorithm,
  $w > log(n)$
- All normal (arithmetical/logical) computations are performed on a Word and they take O(1) time.
- Words can be accessed Randomly. (Random Access Memory).
- Computational Times for many problems can be improved in this model.
  e.g Sorting has been shown to take $O(n\sqrt{loglogn})$ time.

Assumptions in a Word RAM model

- Memory is organized into words of size w.
  A word is 32 bits, or 64 bits in modern day computers.
- If n is the maximum size of the input to the algorithm, $w > log(n)$
- All normal (arithmetical/logical) computations are performed on a Word and they take O(1) time.
- Words can be accessed Randomly. (Random Access Memory).
- Computational Times for many problems can be improved in this model.
  e.g Sorting has been shown to take $O(n\sqrt{loglogn})$ time.

Assumptions in a Word RAM model

- Memory is organized into words of size w.
  A word is 32 bits, or 64 bits in modern day computers.
- If n is the maximum size of the input to the algorithm,
  $w > log(n)$
- All normal (arithmetical/logical) computations are performed on a Word and they take O(1) time.
- Words can be accessed Randomly. (Random Access Memory).
- Computational Times for many problems can be improved in this model.
  e.g Sorting has been shown to take $O(n\sqrt{loglogn})$ time.

Assumptions in a Word RAM model

- Memory is organized into words of size w.
  A word is 32 bits, or 64 bits in modern day computers.
- If n is the maximum size of the input to the algorithm,
  $w > log(n)$
- All normal (arithmetical/logical) computations are performed on a Word and they take O(1) time.
- Words can be accessed Randomly. (Random Access Memory).
- Computational Times for many problems can be improved in this model.
  e.g Sorting has been shown to take $O(n\sqrt{loglogn})$ time.

Outline

Recall the Counting Inversion problem

- We can count the number of Inversions in $O(nlogn)$ time. e.g using Merge Sort.
- But we don't want the actual inversion pairs, only their count. Can something better be done?
- Counting inversions can be reduced to "Dominance Counting" problem - how many points does each point dominate? Use $(i, -\pi(i))$ to map from the set π.
- It has been shown that this can be done in $O(nlogn/loglogn)$ time. (Dietz's data structure).

Recall the Counting Inversion problem

- We can count the number of Inversions in $O(nlogn)$ time. e.g using Merge Sort.
- But we don't want the actual inversion pairs, only their count. Can something better be done?
- Counting inversions can be reduced to "Dominance Counting" problem - how many points does each point dominate? Use $(i, -\pi(i))$ to map from the set π.
- It has been shown that this can be done in $O(nlogn/loglogn)$ time. (Dietz's data structure).

Recall the Counting Inversion problem

- We can count the number of Inversions in $O(nlogn)$ time. e.g using Merge Sort.
- But we don't want the actual inversion pairs, only their count. Can something better be done?
- Counting inversions can be reduced to "Dominance Counting" problem - how many points does each point dominate?
  Use $(i, -\pi(i))$ to map from the set π.
- It has been shown that this can be done in $O(nlogn/loglogn)$ time. (Dietz's data structure).

Recall the Counting Inversion problem

- We can count the number of Inversions in $O(nlogn)$ time. e.g using Merge Sort.
- But we don't want the actual inversion pairs, only their count. Can something better be done?
- Counting inversions can be reduced to "Dominance Counting" problem - how many points does each point dominate? Use $(i, -\pi(i))$ to map from the set π.
- It has been shown that this can be done in $O(nlogn/loglogn)$ time. (Dietz's data structure).

Outline

Partition the input

1. Partition the input into two - those that begin with 0, and those that begin with 1
2. For each element that begins with 0, count how many preceeding elements which start with 1. Add to inversion count.
3. Recursively do this for each of L bits in order.

- If B is the number of words per page, we can do Step 2 in $O(n/B)$ I/O operations.
  Operating Systems move around memory in terms of pages.
- So, for inputs L bits long, we need $O(nL/B)$ I/O operations.

Handling B elements in constant time

- Choose a page size such that the number of words in it $B = w/L$
  In Linux, the standard is 4KB page size. so, on a 64-bit machine, we can have input upto 36-bit numbers. Numbers as big as 4503599627370496.
- The running time becomes $O(nL/B) = O(nL^2/w)$
- For $w \approx logn$, we can simulate word operations in constant time by table lookup.
- The running time becomes linear if $L \approx \sqrt{logn}$
- This word-packing idea is key to speeding up in offline algorithms, as opposed to online algorithms.

An $O(n\sqrt{logn})$algorithm

- How do we solve the original problem with $logn$ bits?
    1. Consider a trie (prefix tree) of depth $(logn)/L$ over the alphabet $[0...2^L]$
       Each node is associated with the elements of the permutation that fall under that node.
    2. For a given node in the trie, the first letters after the common prefix associated with node are L-bit numbers.
    3. Use the above subroutine to compute the number of inversions in this sequence. Add to the running count.
    4. Recurse into each child of the node.
- Each trie can be built in $O(n)$ time per level by bucketing. For $L \approx \sqrt{logn}$, subroutine costs $O(n)$
- Since depth is $(logn)/L$, we get $O(n\sqrt{logn})$ time complexity.

Summary

- Algorithms have different complexities under different computational models.
- Non standard bounds of time complexity can arise in these conditions.
- For realistic computational models lookup tables can help speed up the algorithm if used carefully.

Summary

- Algorithms have different complexities under different computational models.
- Non standard bounds of time complexity can arise in these conditions.
- For realistic computational models lookup tables can help speed up the algorithm if used carefully.

Summary

- Algorithms have different complexities under different computational models.
- Non standard bounds of time complexity can arise in these conditions.
- For realistic computational models lookup tables can help speed up the algorithm if used carefully.