Experimental evaluation of Navigation piles

By
Claus Jensen
University of Copenhagen

Based on the article

NAVIGATION PILES WITH APPLICATIONS TO SORTING, PRIORITY QUEUES, AND PRIORITY DEQUES

By

JYRKI KATAJAINEN and FABIO VITALE

Structure of static navigation piles

- Shape
  - It is a left-complete binary tree of size N, where N ≥ n (# of elements stored).
- Leaf
  - The leaves are used to store the elements.
  - The first n leaves store one element each, the rest are empty.

Structure of static navigation piles

- Branch:
  - A branch (node) contains the index of the leaf storing the top element among the group of elements attached to the leaves in the subtree rooted by the branch.

Structure of static navigation piles

- Representation:
  - The elements are stored in sequence A[0..n).
  - The branches are stored in sequence B[0..N).

Structure of static navigation piles

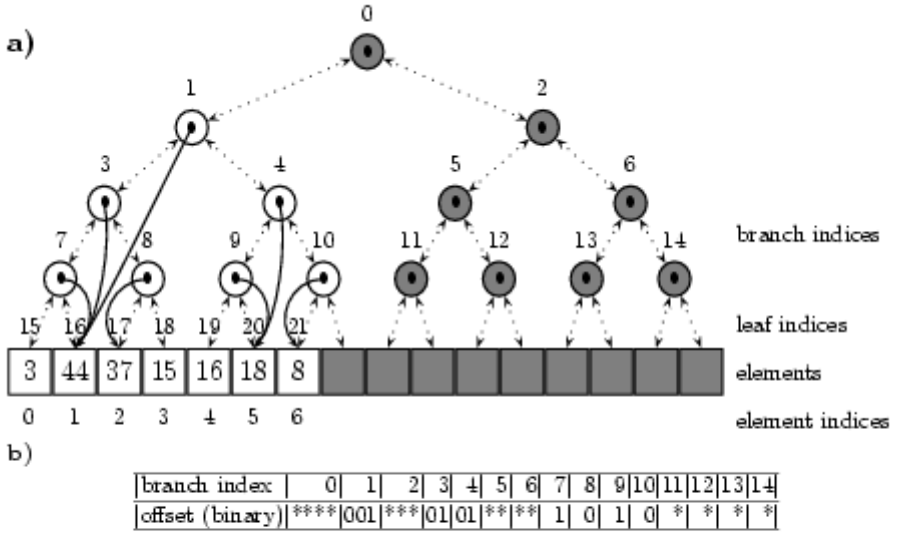


Figure 1. a) A navigation pile of size 7 and capacity 16. The normal parent/child relationships are shown with dotted arrows and the references indicated by the offsets with solid arrows. The gray nodes are not in use. b) The bit representation of the navigation information. Bits marked with * are not in use.

Algorithms

- Make
  - Construction of a navigation pile is done by visiting the branches in a bottom-up manner and computing its index using the indices available at the children.
    - n-1 element comparisons
    - n element moves
    - O(n) instructions

Algorithms

- Push
  - A new element is inserted into the first empty leaf, followed by an update of the branches from the leaf to the root. A binary search strategy can be used in the update.
    - loglog n + O(1) element comparisons
    - 1 element move
    - O(log n) instructions

Algorithms

- Top
  - A reference to the top element stored at the root is returned.
    - O(1) instructions

Algorithms

- Pop
  - The top element is erased and the indices are updated.
  - The update is done in three different ways depending on circumstances inside the navigation pile.
    - Ceiling of log n element comparisons
    - 1 or 2 element moves
    - O(log n) instructions

Algorithms (Pop)

l: element index of the last leaf.

m: element index of the top element.

i: Branch index of the first ancestor of the last leaf which has two children.

j: The element index referred to by the first child of i.

k: The element index referred to by i.

Algorithms (Pop)

- Case 1: m = l
- The top element is erased and the index values on the path from the new last leaf to the root are updated.

Algorithms (Pop)

- Case 2: m ≠ l and k ≠ l
- A[m] ← A[l] and the element stored at the last leaf is erased.
  - The content of the branches on the path from the leaf corresponding to the position m up to the root are updated.

Algorithms (Pop)

- Case 3: m ≠ l and k = l
  - A[m] ← A[j], A[j] ← A[l] and the element stored at the last leaf is erased.
  - The content of the branches on the path [i, root] are updated with a reference to j, if they earlier referred to the last leaf.
  - The content of the branches on the path from the leaf corresponding to the position m up to the root are updated.

Programs

- The navigation pile data structure was implemented in three different versions.
  - The first version used an STL vector to store the indices at the branches.
  - The second version used pointers instead of indices in an attempt to optimize the performance.

Programs

- The third version packed the offsets as suggested in the original article. Offsets indicate the leaf position of the top element within the subtree rooted by the branch. The use of offsets instead of indices reduces the space required to 2N bits, where N is the capacity of the navigation pile.
- All versions were repeatedly improved based on profiling results.

Benchmarks

- Environment A
  - Intel computer:
  - CPU: Pentium 3 (1GHz)
  - Main Memory: RAM 384 MB
  - Cache level 2: 246 KB
  - Operation system: Debian Sarge
  - C++ Compiler: g++ 3.2

Benchmarks

- Environment B
  - AMD computer:
  - CPU: Athlon XP 2100+ (1.73 GHz)
  - Main Memory: RAM 256 MB
  - Cache level 2: 256 KB
  - Operation system: Red Hat 8.0
  - C++ Compiler: g++ 3.2

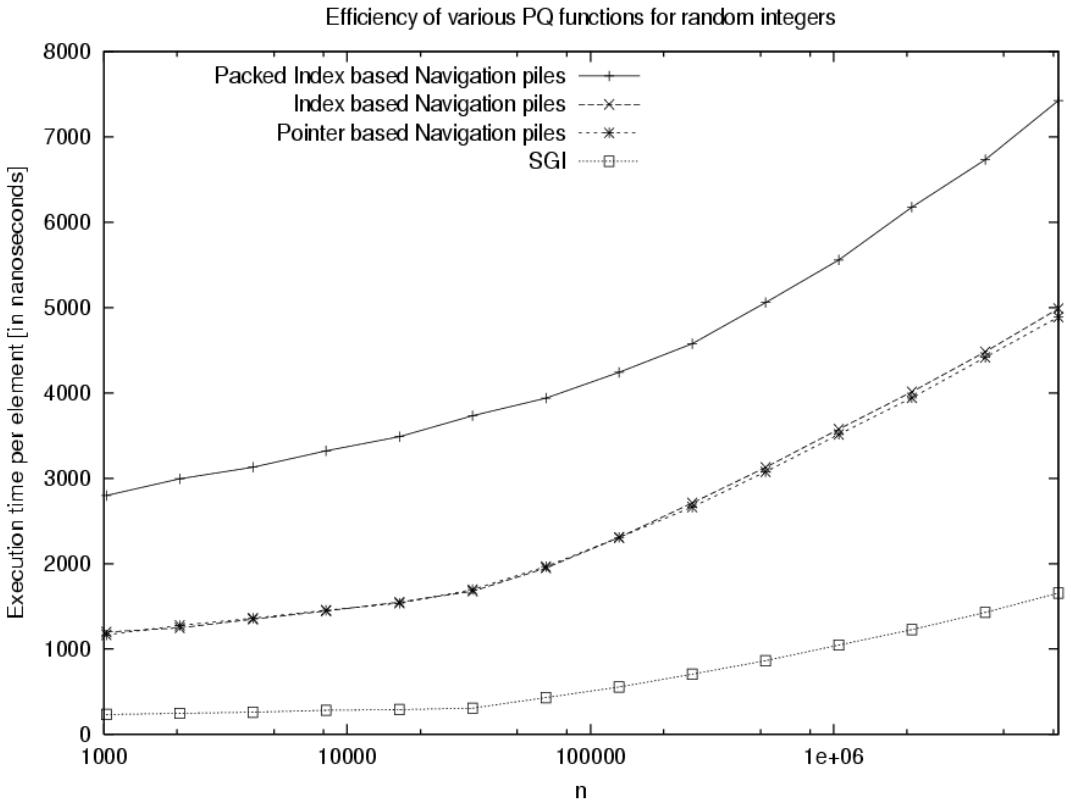
Benchmarks (input data)

	cheap move	expensive move
cheap comparison	unsigned int	big int
expensive comparison	unsigned int ln comparison	(int, big int) ln comparison

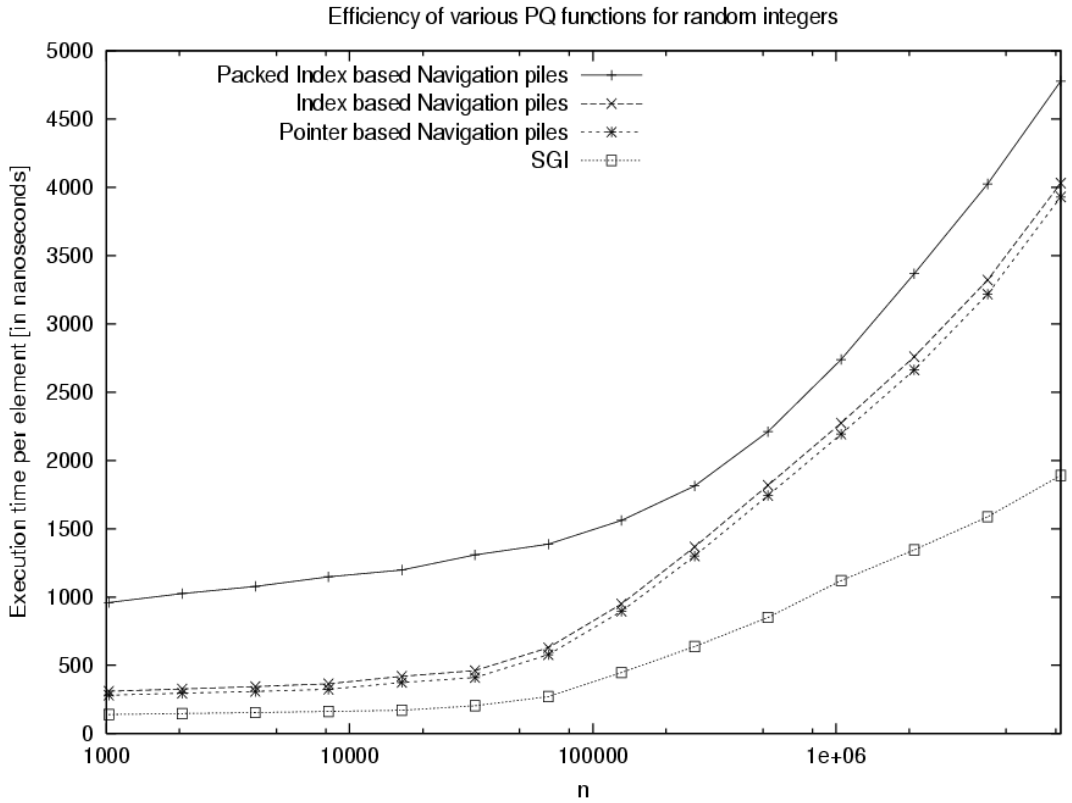
Benchmarks

- Navigation piles were compared to SGI STL's binary heaps.
- The benchmarks performed a construction operation followed by a series of pops.
- The CPH STL benchmark tool was use in the production of the benchmarks.

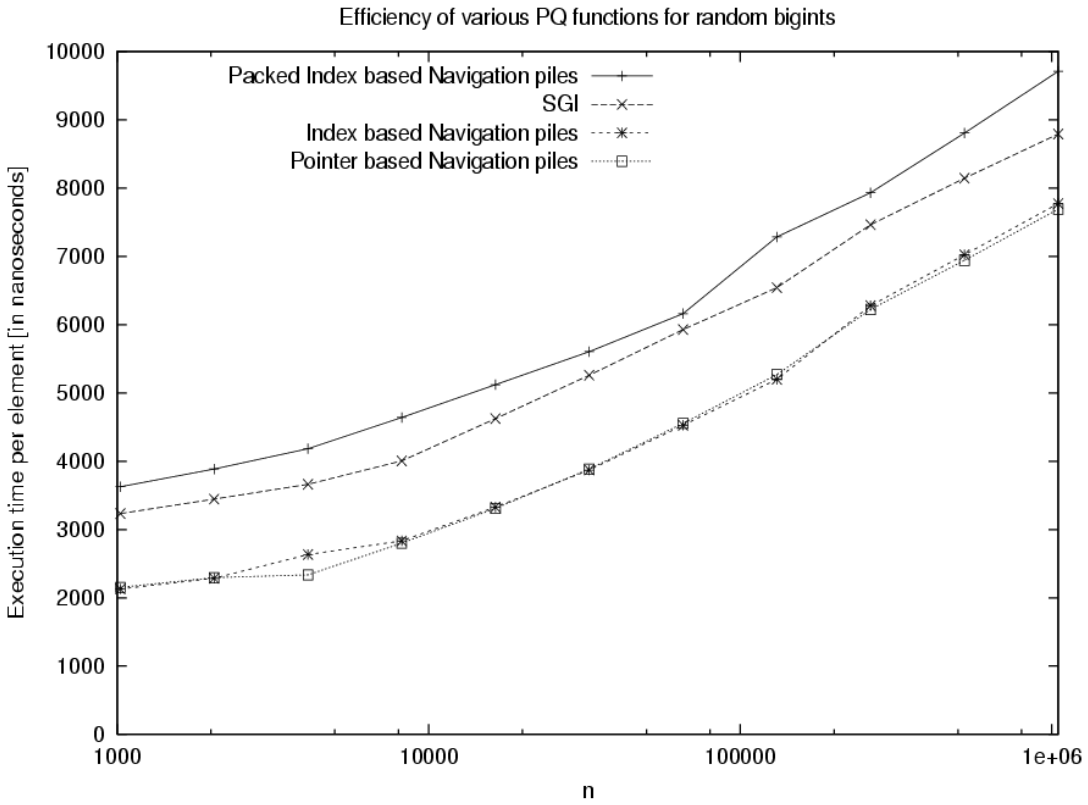
Benchmarks (Intel P3)



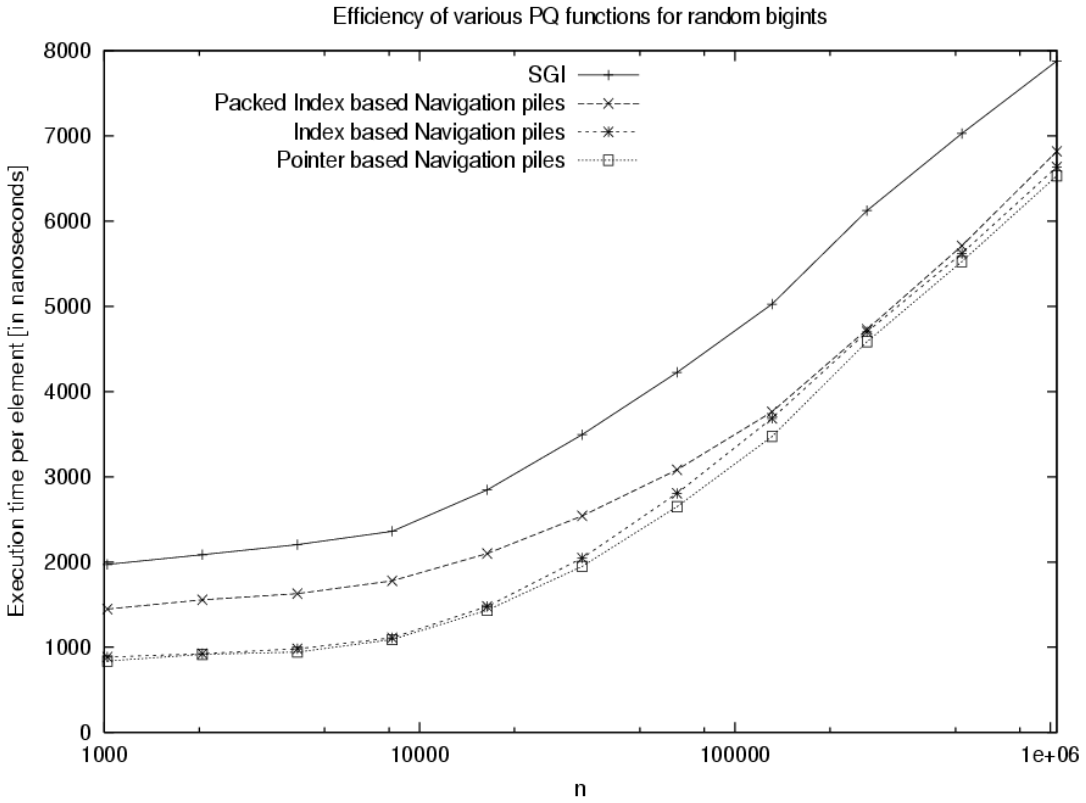
Benchmarks (AMD Athlon)



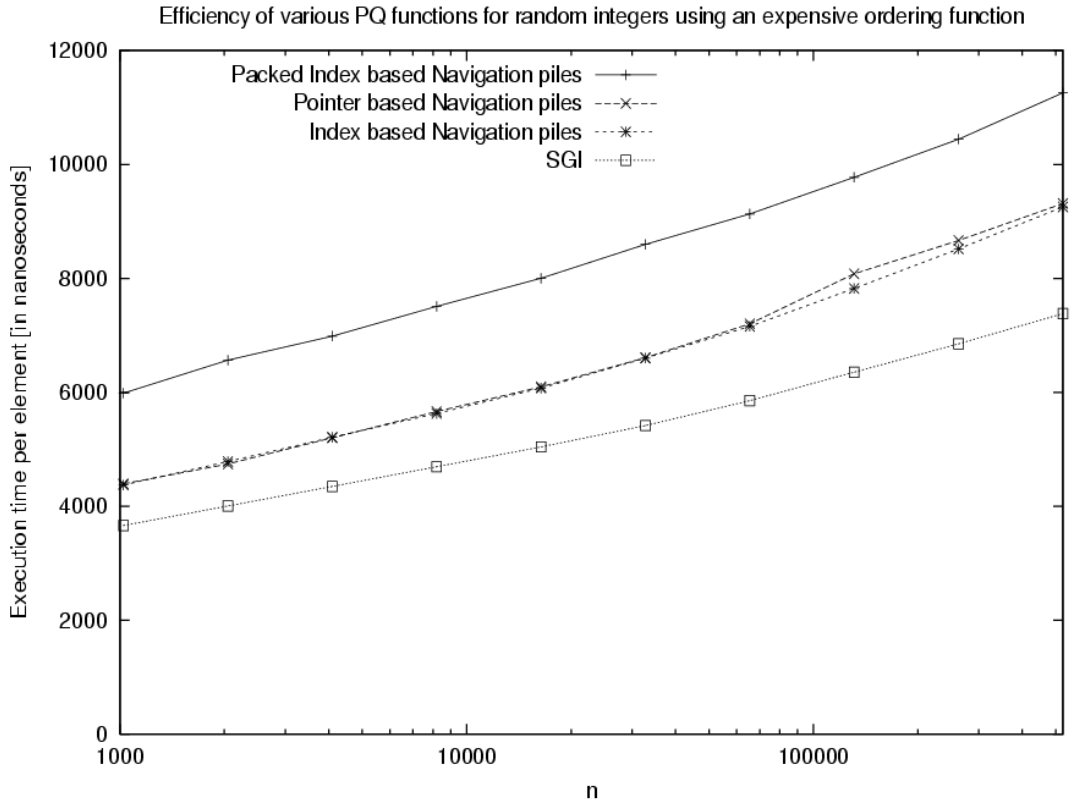
Benchmarks (Intel P3)



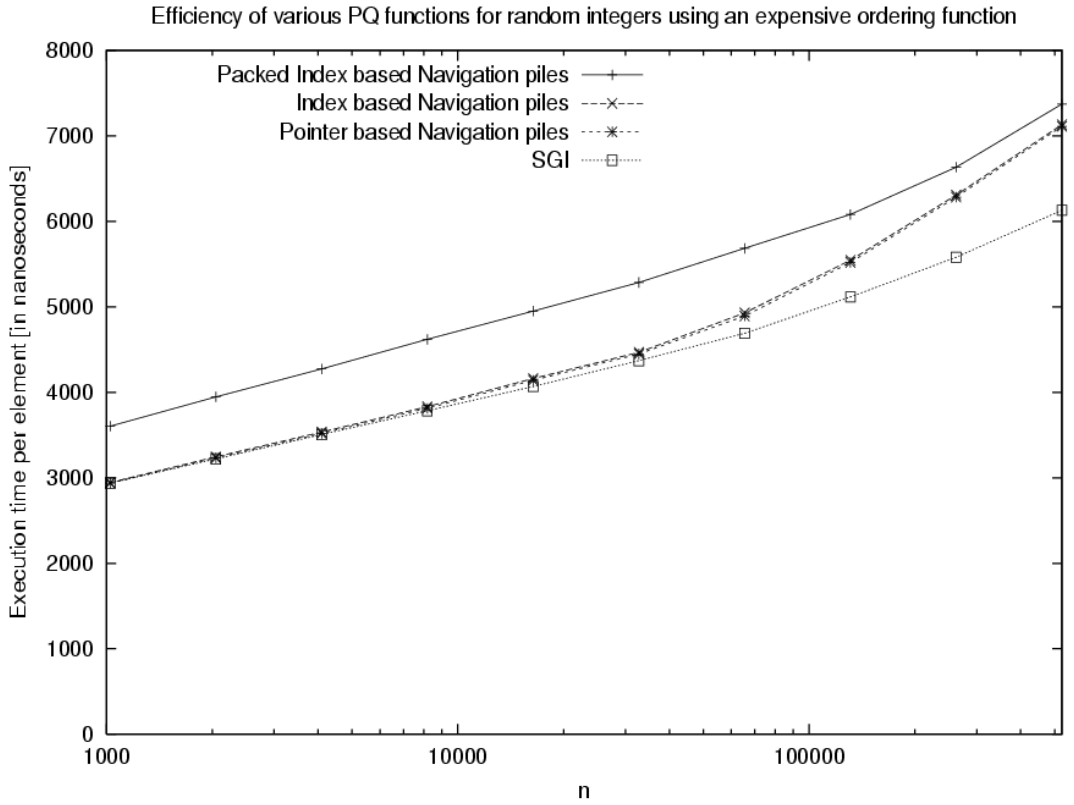
Benchmarks (AMD Athlon)



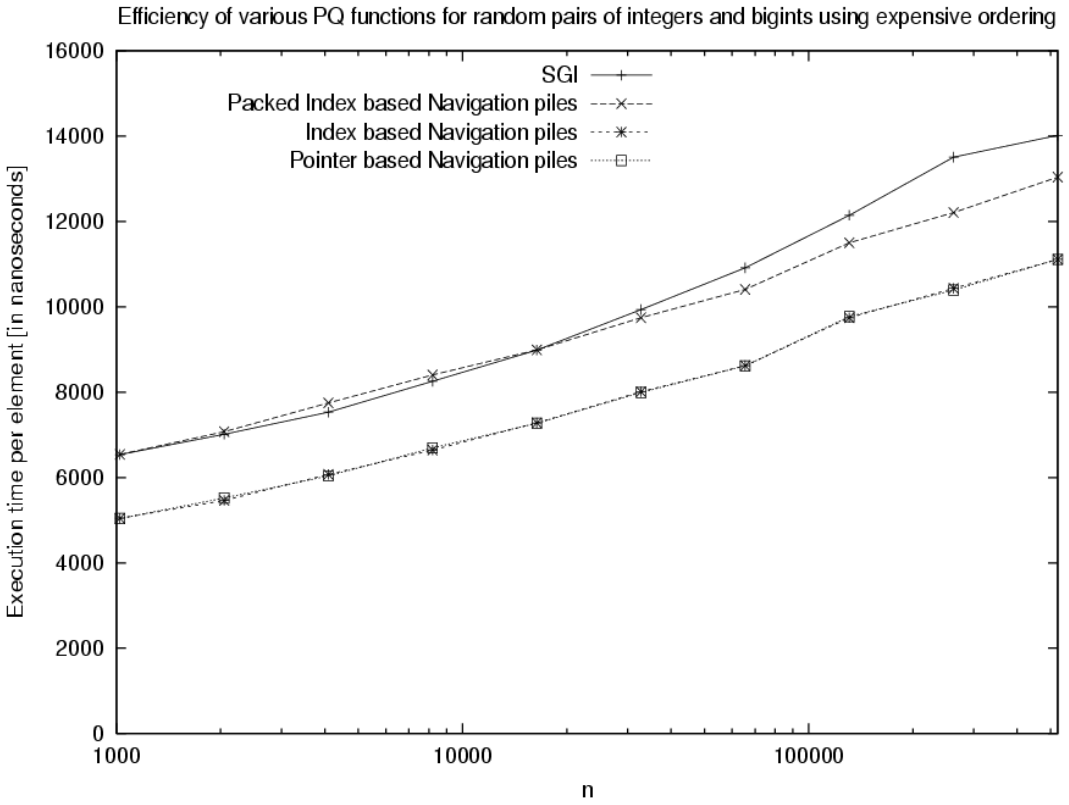
Benchmarks (Intel P3)



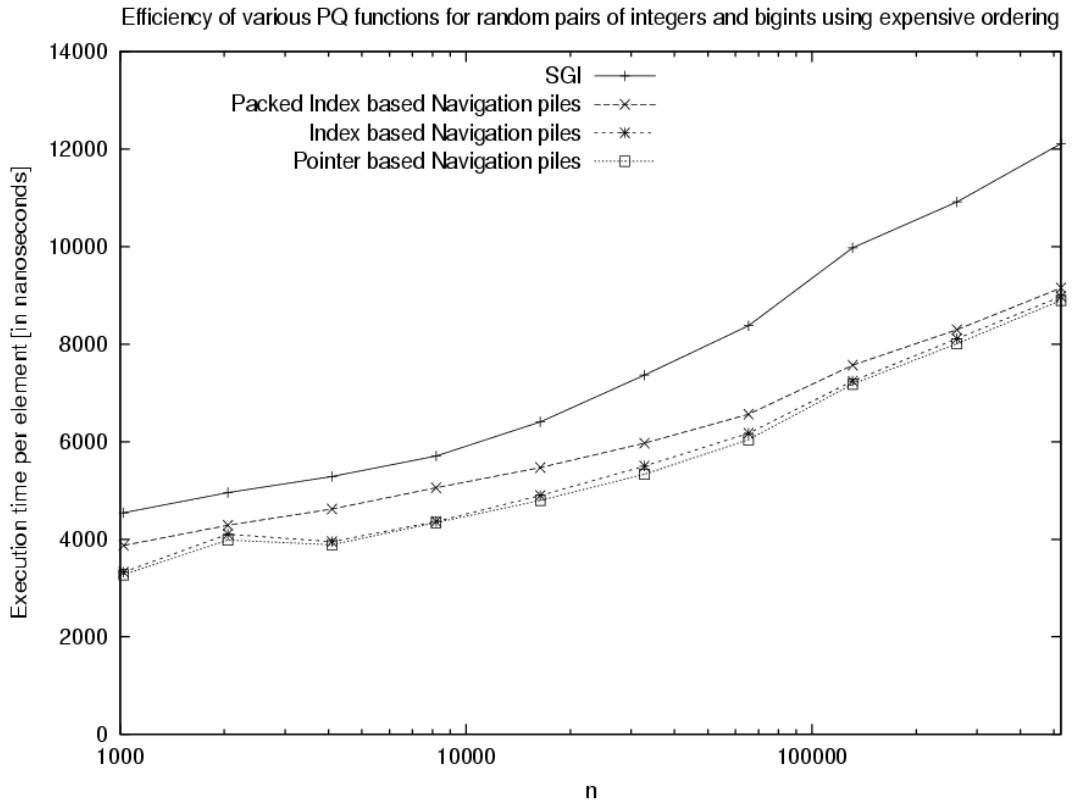
Benchmarks (AMD Athlon)



Benchmarks (Intel P3)



Benchmarks (AMD Athlon)



Questions