Optimal Incremental Sorting

Rodrigo Paredes - Gonzalo Navarro

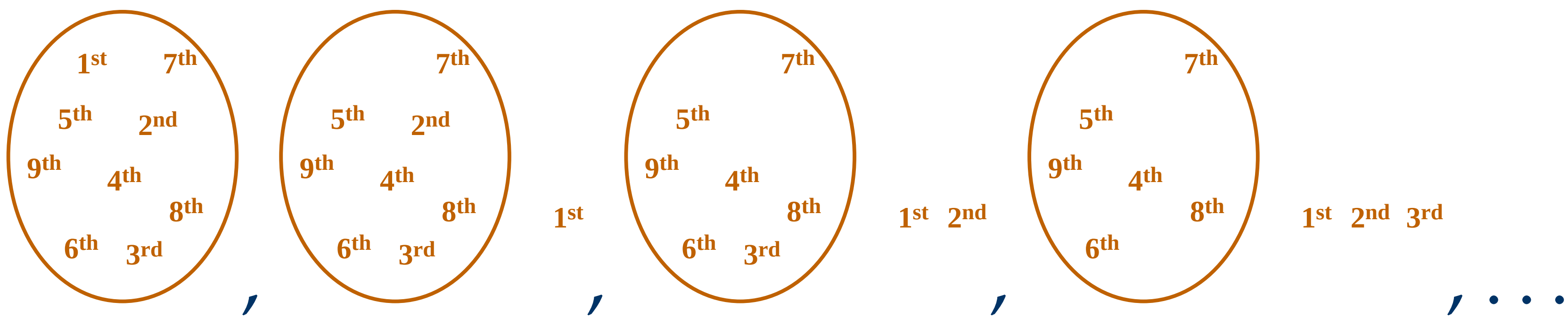
{raparede,gnavarro}@dcc.uchile.cl

Center for Web Research, Department of Computer Science,
Universidad de Chile

Contents

Introduction

- Assume we have a fixed unsorted set A
- Suppose that someone ask for the first object
- and now, for the second, for the third, ..., for the who-knows-th element



- In this scenario we don't know when to stop the incremental searching!!

Algorithmic examples

- Reviewing the edge set one by one while computing the MST with Kruskal's algorithm
- Ranking and returning the most relevant documents from the query outcome chunk by chunk in a Web search engine

Contents

Contents

Basic concepts

The naive solution

- Sort the set A
- Return as many elements as they are required
- But, have we to sort the whole set A to retrieve few elements?

Problem formulation

- *Incremental sorting*

  "Given set A of m numbers, output the elements of A from smallest to largest, so that the process can be stopped after k elements have been output, for any $k \leq m$ that is unknown to the algorithm"

- This is a sort of online problem as k isn't known beforehand

Related work 1

- The offline version is known as *Partial Sorting*:

  "Given set A of m numbers and a integer $k \leq m$, output the smallest k elements of A in ascending order"

- Asymptotical optimal solution:
  - Select p, which is the k-th element of A, with QUICKSELECT in time $O(m)$ in average
  - Sorting the elements in the left partition, those ones smaller than p, with QUICKSORT in time $O(k \log k)$ in average

Related work 2

- Total time $O(m + k \log k) \leq O(m \log m)$
- Partial Quick Sort (PQS) interleaves the QUICKSELECT and QUICKSORT stages saving a bit of work
- Current online solution: Minheapify the array, and minima extractions from the heap (HEx)

Contents

Contents

IncrementalQuickSelect

- IncrementalQuickSelect (IQS) is an $O(m + k \log k)$ time on average algorithm to solve the Incremental sorting problem
- Applications:
  - Kruskal's MST Algorithm
  - Ranking a set
  - Partial sorting from any place of the array
  - Priority Queues (!?)

A wrong way

- Call QUICKSELECT on $A[0, m-1]$
- Call QUICKSELECT on $A[1, m-1]$
- $\ldots$
- Call QUICKSELECT on $A[k-1, m-1]$
- The overall complexity is $O(km)$ $(> O(m + k \log k))$, and $A[0, k-1]$ is sorted,
- But what's wrong?

The Core Idea

When we call QUICKSELECT on $A[1, m-1]$, we have already compute a decreasing sequence of pivots from the invocation on $A[0, m-1]$.

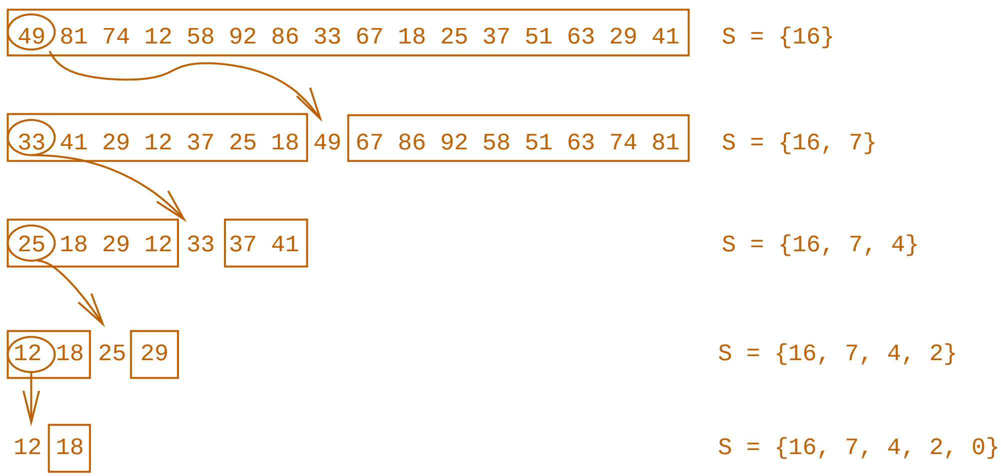
```
49 81 74 12 58 92 86 33 67 18 25 37 51 63 29 41      S = {16}

33 41 29 12 37 25 18 49 67 86 92 58 51 63 74 81      S = {16, 7}

25 18 29 12 33 37 41                                 S = {16, 7, 4}

12 18 25 29                                          S = {16, 7, 4, 2}

12 18                                                S = {16, 7, 4, 2, 0}
```

Reuse the computed pivots!!

Using the stored pivots

Given the pivots in S, the next minimum place is:

- The index on top of S
- $(A[0] < A[1, S.\mathrm{top}() - 1] < A[S.\mathrm{top}(), m] \Rightarrow)$ QUICKSELECT$(A[1, S.\mathrm{top}() - 1])$, and store the next pivots



place	value	obs
2^{nd}	18	chunk
3^{rd}	25	pivot
4^{th}	29	chunk
5^{th}	33	pivot

The algorithm

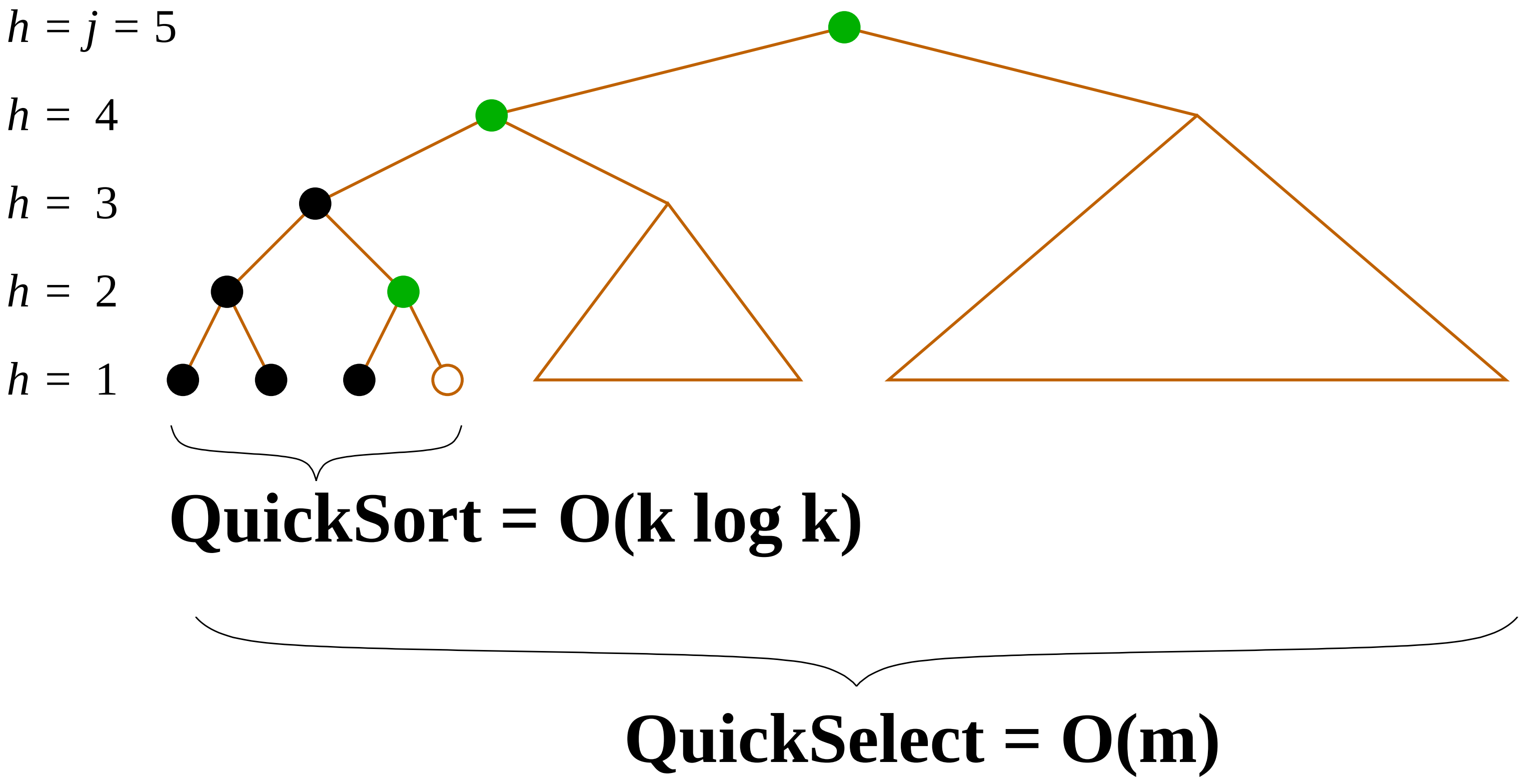
$$
\begin{array}{l}
\textbf{IQS}\ (\text{Set } A, \text{Int } idx, \text{Stack } S) \\
\quad \textbf{If } idx = S.\text{top}()\ \textbf{Then } S.\text{pop}(),\ \textbf{Return } A[idx] \\
\quad pidx \leftarrow \textbf{random}[idx, S.\text{top}()-1] \\
\quad pidx' \leftarrow \textbf{partition}(A, A[pidx], idx, S.\text{top}()-1) \\
\quad //\ A[0] \le \ldots \le A[idx-1] \\
\quad //\ \le A[idx, pidx'-1] \le A[pidx'] \\
\quad // \qquad\qquad\qquad\qquad \le A[pidx'+1, S.\text{top}()-1] \\
\quad //\ \le A[S.\text{top}(), m-1] \\
\quad S.\text{push}(pidx') \\
\quad \textbf{Return IQS}(A, idx, S)
\end{array}
$$

Contents

- Introduction
- Basic concepts
- IncrementalQuickSelect
- Sketch of IQS complexity
- IQS and Kruskal's MST
- Experimental results
- Conclusions

Contents

Complexity sketch



The recurrence

$$T(m,k) = m - 1 + \frac{1}{m}\left(\sum_{p=k}^{m-1} T(p,k) \right.$$
$$+ T(k-1,k-1)$$
$$\left. + \sum_{p=0}^{k-2}\Big(T(p,p) + T(m-1-p, k-p-1)\Big)\right)$$

$$T(m,k) \ < \ 4m - 8k + (3k+1)H_k \ < \ 4m + 3kH_k$$

Contents

Contents

Kruskal's MST

Kruskal1 (Graph $G(V, E)$)

UnionFind $C \leftarrow \{v \in V, \{v\}\}$
// the set of all connected components
$mst \leftarrow \emptyset$ // the growing minimum spanning tr
ascendingSort(E), $k \leftarrow 0$
While $|C| > 1$ **Do**
// select an edge in ascending order
$(e = \{u, v\}) \leftarrow E[k], k \leftarrow k + 1$
If $C.$**find**$(u) \neq C.$**find**(v) **Then**
$mst \leftarrow mst \cup \{e\}$, $C.$**union**(u, v)
Return mst

IQS and Kruskal's MST

Kruskal3 (Graph $G(V, E)$)

  UnionFind $C \leftarrow \{v \in V, \{v\}\}$

  // the set of all connected components

  $mst \leftarrow \emptyset$ // the growing minimum spanning tr

  Stack S, S.**push**(m), $k \leftarrow 0$ // $m = |E|$

  While $|\mathrm{C}| > 1$ **Do**

    // select the lowest edge incrementally

    $(e = \{u, v\}) \leftarrow$ **IQS**(E, k, S), $k \leftarrow k + 1$

    If C.**find**$(u) \neq$ C.**find**(v) **Then**

      $mst \leftarrow mst \cup \{e\}$, C.**union**$(u, v)$

  Return mst

Contents

Contents

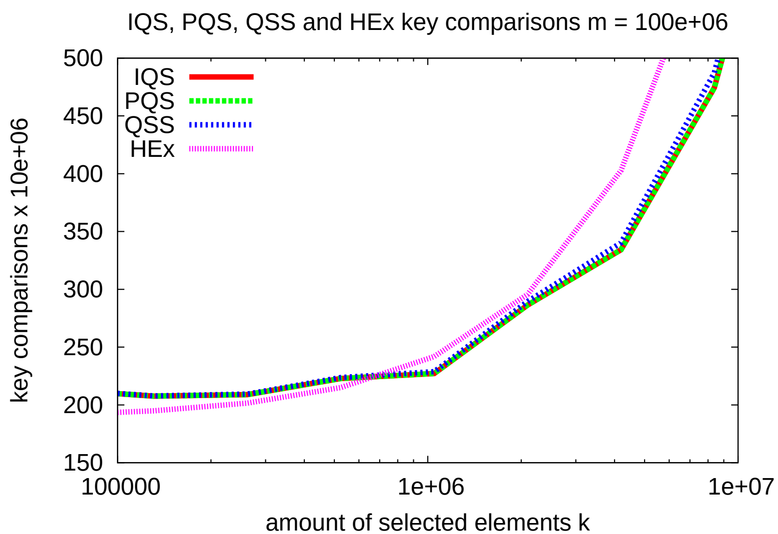
Experimental results

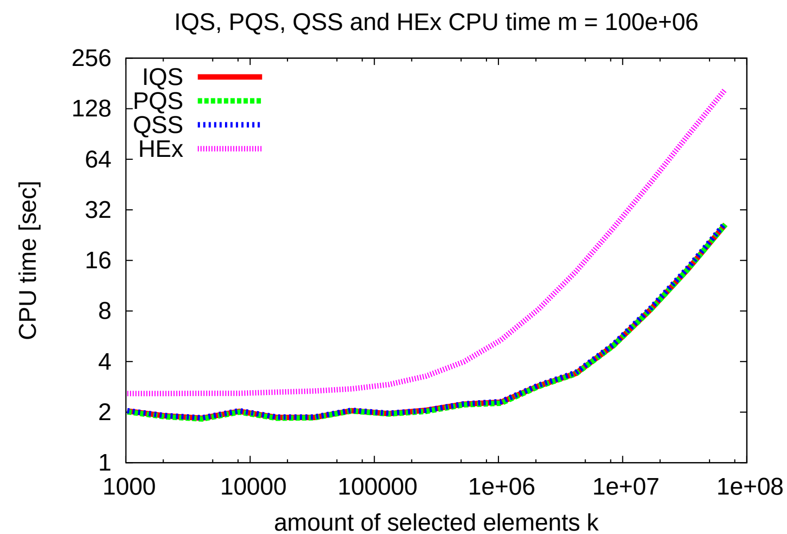
- Evaluating IQS
- Evaluating Kruskal3

Evaluating IQS 1

	Fitting	Error
$\textbf{PQS}_{cpu}$	$19.70m + 14.21k \log_2 k$	3.90%
$\textbf{PQS}_{cmp}$	$2.047m + 1.301k \log_2 k$	3.55%
$\textbf{IQS}_{cpu}$	$19.88m + 14.21k \log_2 k$	3.89%
$\textbf{IQS}_{cmp}$	$2.047m + 1.301k \log_2 k$	3.55%
$\textbf{QSS}_{cpu}$	$20.00m + 14.52k \log_2 k$	3.89%
$\textbf{QSS}_{cmp}$	$2.050m + 1.362k \log_2 k$	3.61%
$\textbf{HEx}_{cpu}$	$25.96m + 85.88k \log_2 m$	5.05%
$\textbf{HEx}_{cmp}$	$1.892m + 1.875k \log_2 m$	0.65%

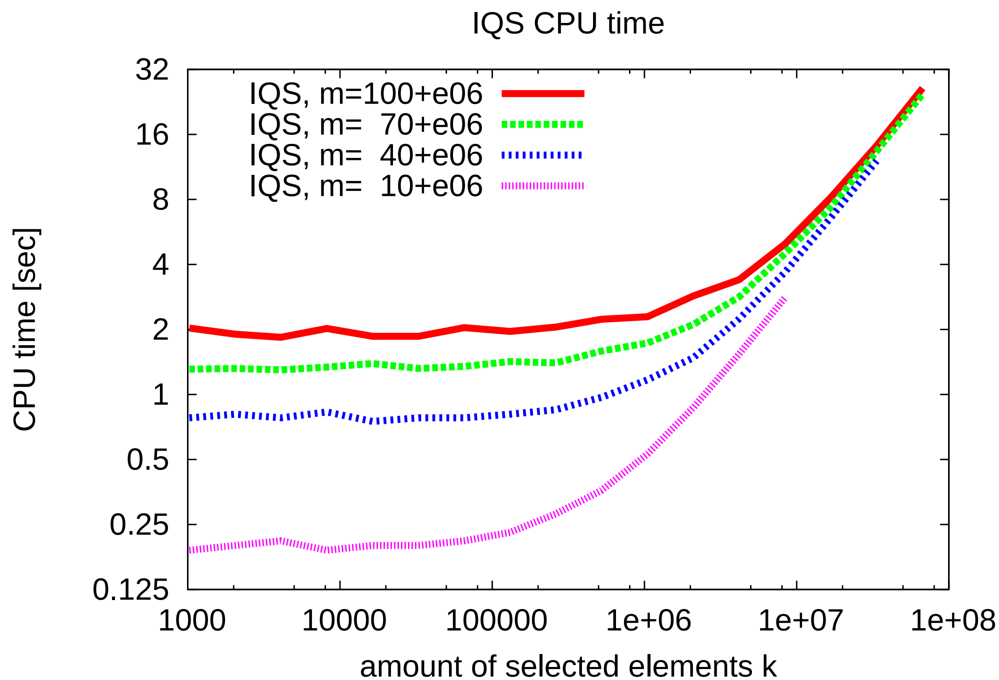
Evaluating IQS 2





- Almost the same time that PQS or QSS
- HEx (the online approach) is 6 times slower than IQS

Evaluating IQS 3



- $k \leq 0.01m$: dominates m
- $0.01m < k \leq 0.04m$: both terms take part in the cost
- $0.04m < k$: dominates $k \log k$

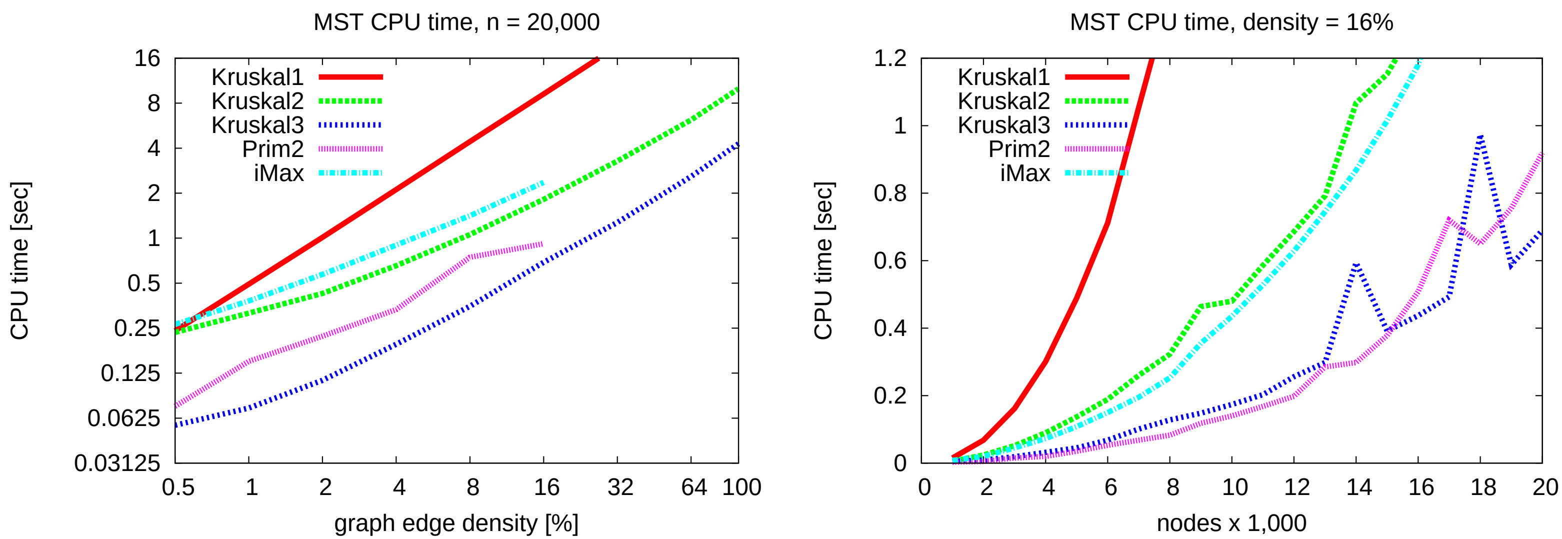
Evaluating Kruskal3 1

	Fitting	Error
Sorted edges	$0.532 n \ln n$	1.47%
Kruskal1$_{cpu}$	$12.23 m \log_2 m$	6.74%
Kruskal2$_{cpu}$	$51.62 m + 34.84 n \log_2 n \log_2 m$	9.62%
Kruskal3$_{cpu}$	$21.67 m + 10.01 n \log_2^2 n$	9.75%

Note that we expect to sort $\frac{1}{2} n \ln n$ edges

[Jason, Knuth, Łucsak and Pittel, *The birth of the giant component*, 1993]

Evaluating Kruskal3 2



- We have the fastest Kruskal's MST implementation, for all edge density, and for all $|V|$:-)

Contents

- Introduction
- Basic concepts
- IncrementalQuickSelect
- Sketch of IQS complexity
- IQS and Kruskal's MST
- Experimental results
- Conclusions

Contents

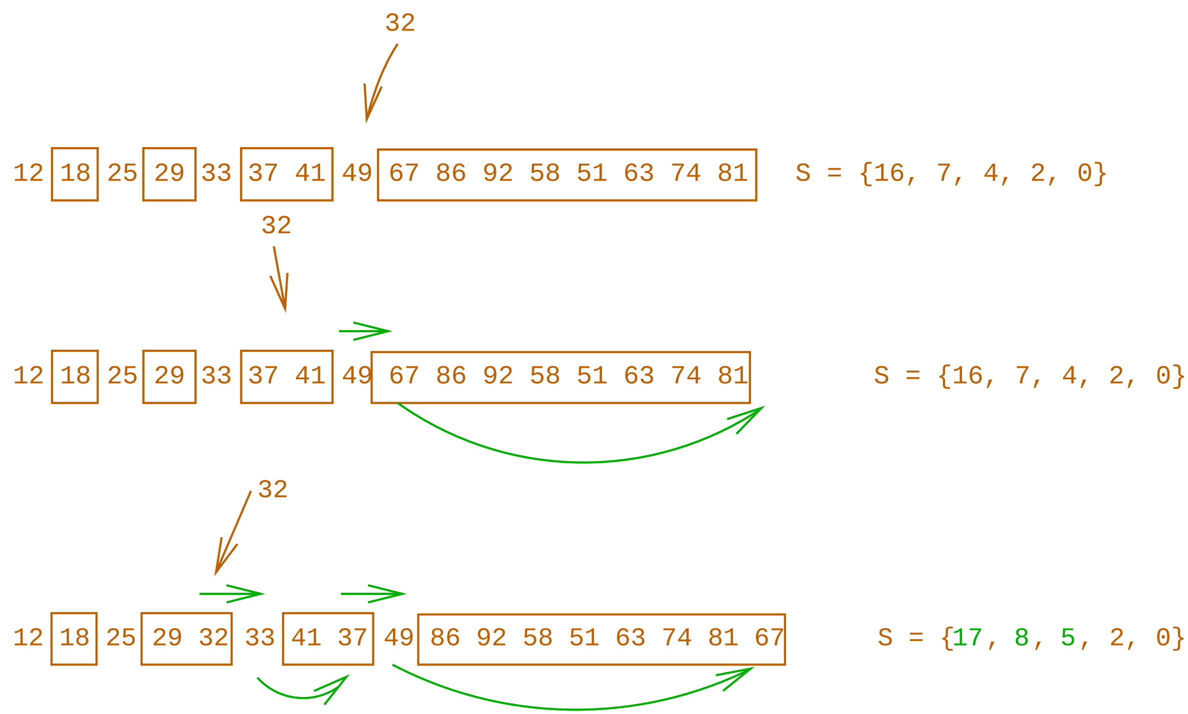
- Introduction
- Basic concepts
- INCREMENTALQUICKSELECT
- Sketch of IQS complexity
- IQS and Kruskal's MST
- Experimental results
- Conclusions

Conclusions

- IQS is an algorithm to incrementally retrieve the next smallest element from a set.
- IQS has the same complexity than existing solutions, but
- IQS as fast as the optimal offline algorithm (PQS)
- IQS have practical applicability in many problems: MST, ranking, partial sorting form any place of the array, and priority queues

Next work: MinQuickHeaps

- Heapify: IQS$(A, 0, S \leftarrow \{|A|\})$: $O(m)$
- Extraction: IQS(A, k, S): $O(\log k)$ amortized
- Insertion: We need to move some pivots



Can we improve the Prim's MST algorithm?

- The *DecreaseKey* also can be performed efficiently
- Worst case $O(\log n)$, but in practice it could be $O(1)$ in average
- We can implement Prim's with minQH, and using the DecreaseKey operation, obtain an $O(m + n \log n)$ MST algorithm :-)

Optimal Incremental Sorting

Rodrigo Paredes - Gonzalo Navarro

{raparede,gnavarro}@dcc.uchile.cl

Center for Web Research, Department of Computer Science,
Universidad de Chile