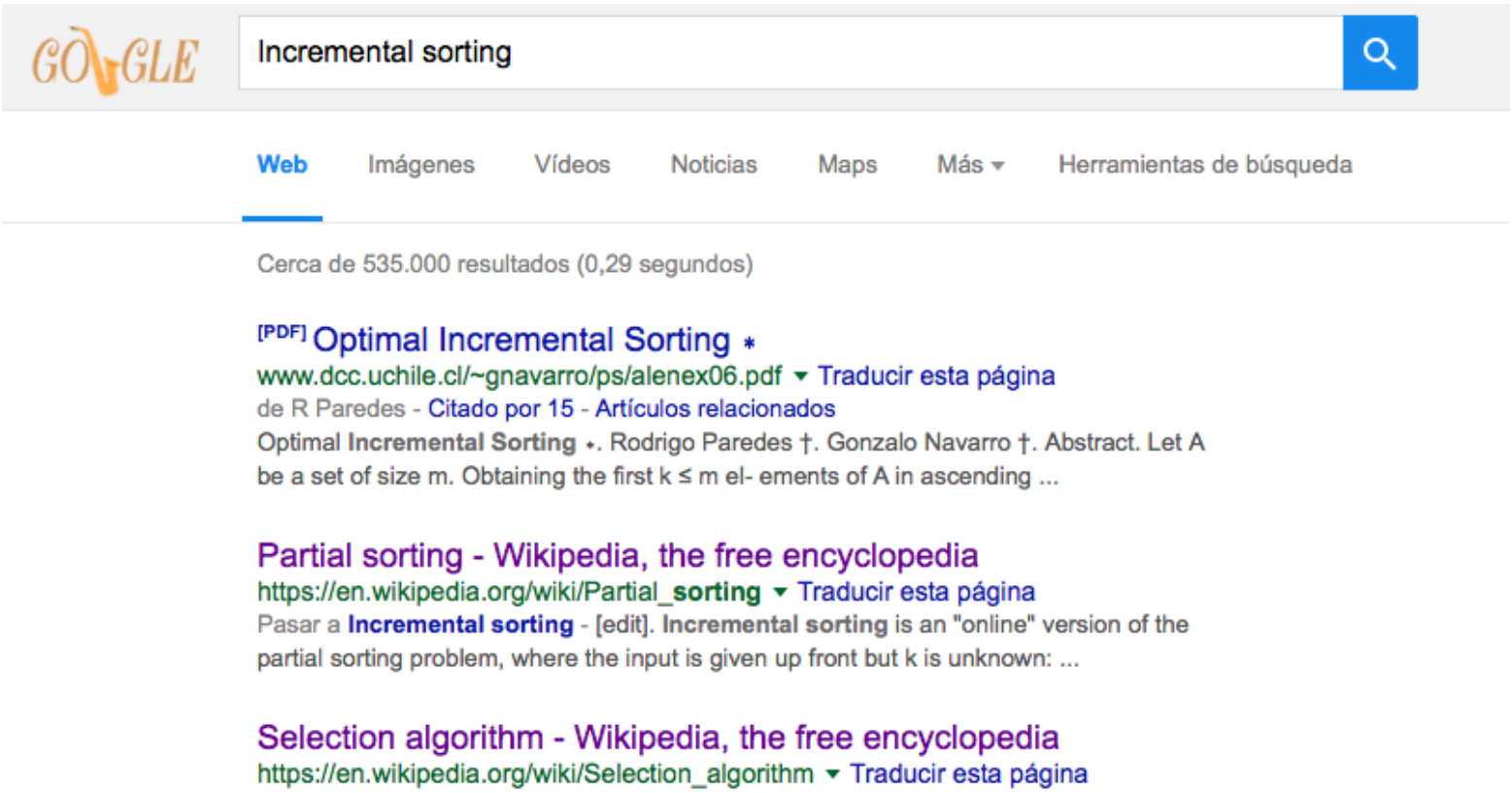
Worst-case Optimal Incremental Sorting

Erik Regla Rodrigo Paredes

Departamento de Ciencias de la Computación
Universidad de Talca

XXXIV International Conference of the
Chilean Computer Science Society (SCCC), 2015


GOOGLE
Incremental sorting
Web
Imágenes
Vídeos
Noticias
Maps
Más
Herramientas de búsqueda
Cerca de 535.000 resultados (0,29 segundos)
[PDF] Optimal Incremental Sorting *
www.dcc.uchile.cl/~gnavarro/ps/alenex06.pdf ▾ Traducir esta página
de R Paredes - Citado por 15 - Artículos relacionados
Optimal Incremental Sorting *. Rodrigo Paredes †. Gonzalo Navarro †. Abstract. Let A be a set of size m. Obtaining the first k ≤ m el- ements of A in ascending ...
Partial sorting - Wikipedia, the free encyclopedia
https://en.wikipedia.org/wiki/Partial_sorting ▾ Traducir esta página
Pasar a Incremental sorting - [edit]. Incremental sorting is an "online" version of the partial sorting problem, where the input is given up front but k is unknown: ...
Selection algorithm - Wikipedia, the free encyclopedia
https://en.wikipedia.org/wiki/Selection_algorithm ▾ Traducir esta página

Incremental Sorting

- For an unsorted set A of size n, we want to retrieve the k smallest elements from smaller to largest in optimum time. $O(n\ + k\ log_2(k))$
- There is no information beforehand the value of k.
- If $k = n$ the query equals to sort the entire array, but if $k < \frac{n}{log_2(n)}$ sorting the entire array is meaningless.

Incremental Quicksort
Overview

- Presented by Paredes and Navarro in 2006 (Optimal Incremental Sorting at ALENEX06).
- Based on Quicksort
- The pivots used on the partition stage of Quicksort are stored on a stack S that later is used to ease future calls.
- Experimental results showed that Incremental Quicksort (IQS) is up to 4 times faster than the classical alternative that uses binary heaps

Incremental Quicksort
Overview

- Presented by Paredes and Navarro in 2006 (Optimal Incremental Sorting at ALENEX06).
- Based on Quicksort
- The pivots used on the partition stage of Quicksort are stored on a stack S that later is used to ease future calls.
- Experimental results showed that Incremental Quicksort (IQS) is up to 4 times faster than the classical alternative that uses binary heaps in expected case.

Incremental Quicksort
Explanation

1: **procedure** IQS(A, S, k)
2: **while** $k < S.top()$ **do**
3: $pidx \leftarrow random(k, S.top() - 1)$
4: $pidx \leftarrow partition(A_{k,S.top()-1}, pidx)$
5: $S.push(pidx)$
6: **end while**
7: $S.pop()$
8: **return** A_k
9: **end procedure**

Figure: Incremental Quicksort. Stack S is initialized as $S \leftarrow |A|$.

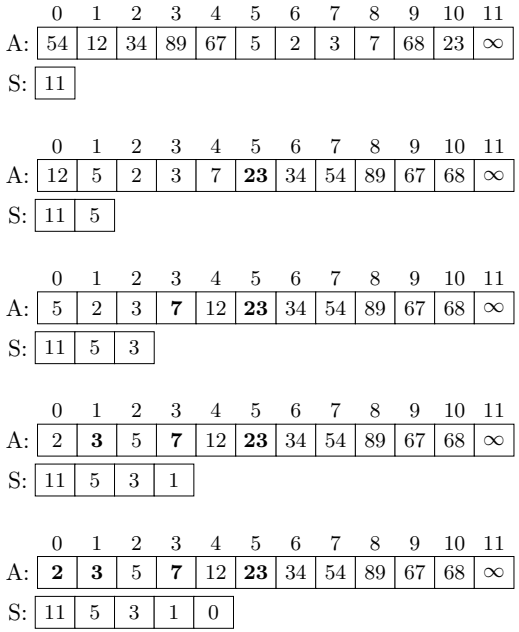


Figure: Example of a normal IQS execution retrieving the first element of A. Pivot selection is fixed as the last element.

Incremental Quicksort
Failure

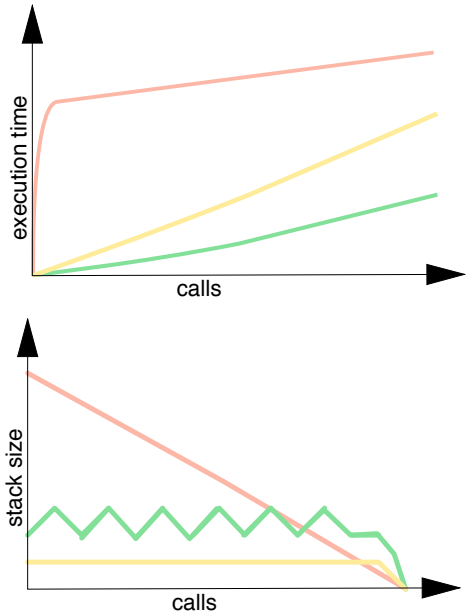


Figure: Graphical example of situations when partitioning with pivots at the left (yellow), in the middle (green) and right side (red) of A.

	0	1	2	3	4	5	6	7	8	9	10	11
A:	2	3	5	7	12	23	34	54	67	68	89	∞
S:	11											

	0	1	2	3	4	5	6	7	8	9	10	11
A:	2	3	5	7	12	23	34	54	67	68	**89**	∞
S:	11	10										

	0	1	2	3	4	5	6	7	8	9	10	11
A:	2	3	5	7	12	23	34	54	67	**68**	**89**	∞
S:	11	10	9									

	0	1	2	3	4	5	6	7	8	9	10	11
A:	2	3	5	7	12	23	34	54	**67**	**68**	**89**	∞
S:	11	10	9	8								

	0	1	2	3	4	5	6	7	8	9	10	11
A:	2	3	5	7	12	23	43	**54**	**67**	**68**	**89**	∞
S:	11	10	9	8	7							

⋮

	0	1	2	3	4	5	6	7	8	9	10	11
A:	**2**	**3**	**5**	**7**	**12**	**23**	**43**	**54**	**67**	**68**	**89**	∞
S:	11	10	9	8	7	6	5	4	3	2	1	0

Figure: Example of a failed IQS execution retrieving the first element because the pivot is always the last element.

Introspective approach to IQS

Musser's Introsort

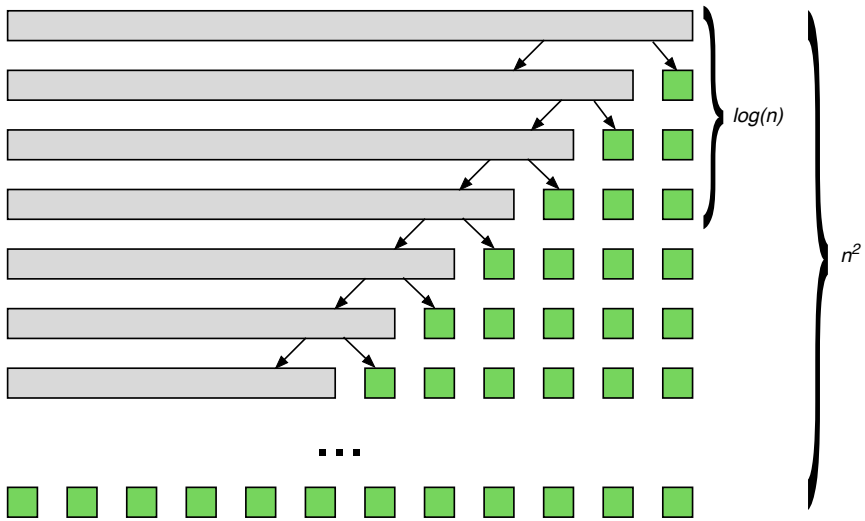


Figure: Example of a failed Quicksort execution. Due to the linear problem reduction the partition stage is executed n times over the array, computational complexity is $O(n^2)$.

Introspective approach to IQS

Musser's Introsort

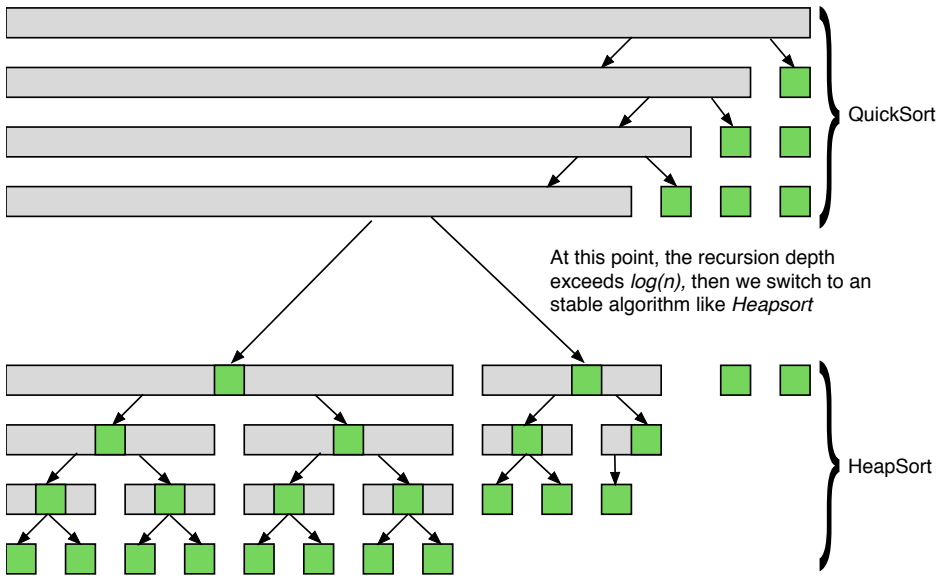


Figure: Example of Introsort execution. When the number of recursions performed by Quicksort excedes $log_2(n)$ calls, Introsort changes to Heapsort which has stable $O(n\ log_2(n))$ computational complexity.

Introspective approach to IQS

Blum et al. Algorithm and Quickselect

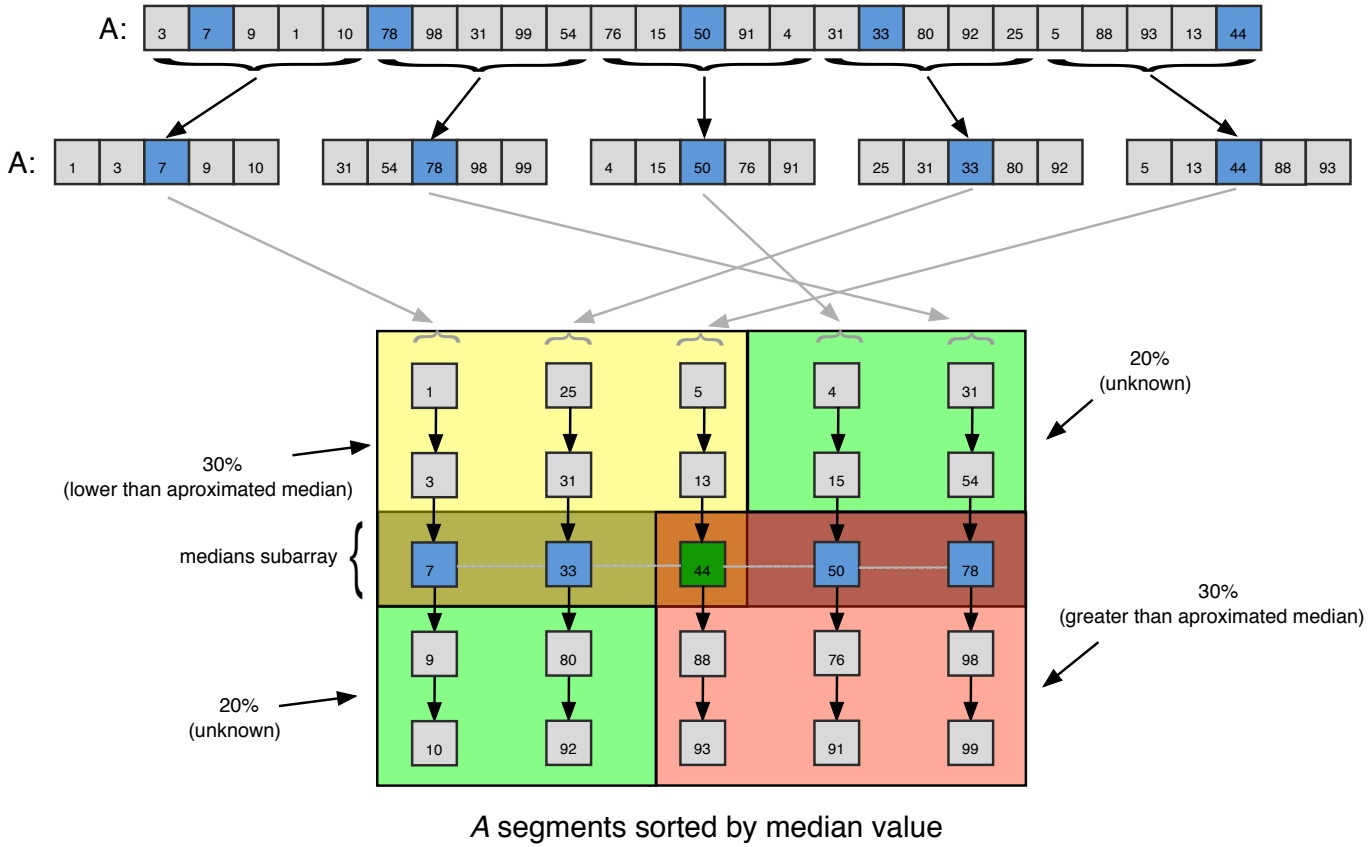
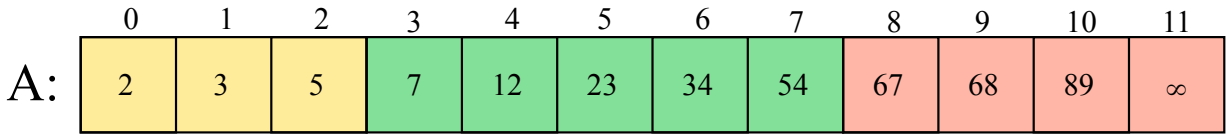


Figure: Example of Blum et al. algorithm execution.

Introspective approach to IQS

Pivot selection



- Pivot belongs to $\mathbb{P}_{30}$ (yellow): Fast query response but does not ease future calls. Stack begins to empty.
- Pivot belongs to $\mathbb{P}_{70}$ but not to $\mathbb{P}_{30}$ (green): Expected pivot in average case and Blum et at. guaranteed median range.
- Pivot does not belong to $\mathbb{P}_{70}$(red): Useless pivot. Stack begins to populate.

Introspective approach to IQS

Issues

- If we wait for the recursion depth to reach a certain value to trigger introspection, since the first queries are the most heavyweight ones, introspection is useless.
- It is not possible to use permanently Blum et al. algorithm because its linear complexity ($T(n) = 11n$ for 5 group selection).

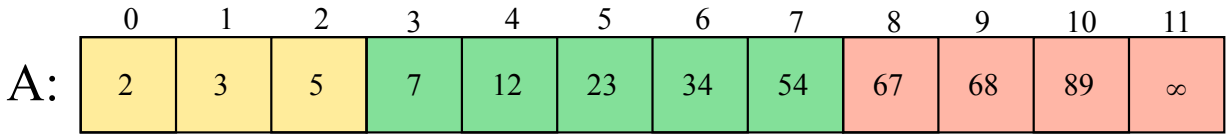
Introspective approach to IQS

Issues

- If we wait for the recursion depth to reach a certain value to trigger introspection, since the first queries are the most heavyweight ones, introspection is useless.
- It is not possible to use permanently Blum et al. algorithm because its linear complexity ($T(n) = 11n$ for 5 group selection).
- We cannot change IQS behavior by changing its selection method, since the stack controls how the recursions are made it is feasible to control which data it stores.

Introspective approach to IQS

Pivot rejection/acceptation



- Pivot belongs to $\mathbb{P}_{30}$ (yellow): Pivot stored on S plus an extra pivot using Blum et al. algorithm.
- Pivot belongs to $\mathbb{P}_{70}$ but not to $\mathbb{P}_{30}$ (green): Pivot is stored. No extra work is needed.
- Pivot does not belong to $\mathbb{P}_{70}$(red): Pivot discarded and replaced by a new one using Blum et al. algorithm.

Introspective Incremental Quicksort

Algorithm

```
1:  procedure IIQS(A, S, k)
2:      while k < S.top() do
3:          pidx ← random(k, S.top() − 1)
4:          pidx ← partition(A_{k,S.top()−1}, pidx)
5:          m ← S.top() − k
6:          α ← 0.3
7:          r ← −1
8:          if pidx < k + αm then
9:              r ← pidx
10:             pidx ← pick(A_{r+1,S.top()−1})
11:             pidx ← partition(A_{r+1,S.top()−1}, pidx)
12:         else if pidx > S.top() − αm then
13:             r ← pidx
14:             pidx ← pick(A_{k,pidx})
15:             pidx ← partition(A_{k,r}, pidx)
16:             r ← −1
17:         end if
18:         S.push(pidx)
19:         if r > −1 then
20:             S.push(r)
21:         end if
22:     end while
23:     S.pop()
24:     return A_k
25: end procedure
```

$$T(n) = \overbrace{T\left(\frac{7}{10}n\right)}^{\text{next iteration}} + \overbrace{n-1}^{\text{partition cost}} + \overbrace{11n}^{\text{introspective step cost}}$$

$$T(1) = 0$$

Using $n = \gamma^k$, where $\gamma = \frac{10}{7}$

$$T(\gamma^k) = T\left(\gamma^{k-1}\right) + 12\gamma^k - 1$$

Using telescopic series we obtain

$$T(\gamma^k) = 12\sum_{i=1}^{k}\gamma^i - k = 12\frac{\gamma(\gamma^k - 1)}{\gamma - 1} - \log_\gamma n$$

Reverting changes to $T(n)$ and evaluating γ

$$T(n) = 40(n-1) - log_{\frac{10}{7}} n$$

$$T(n) = O(n)$$

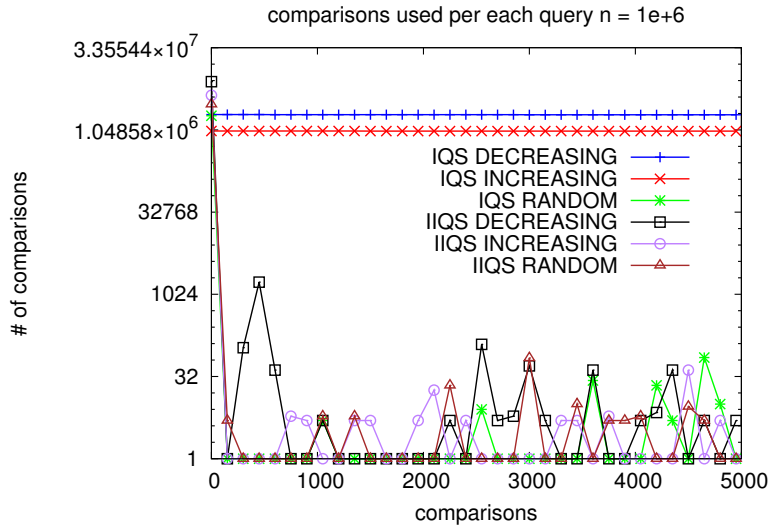
Introspective Incremental Quicksort

Experimental setup

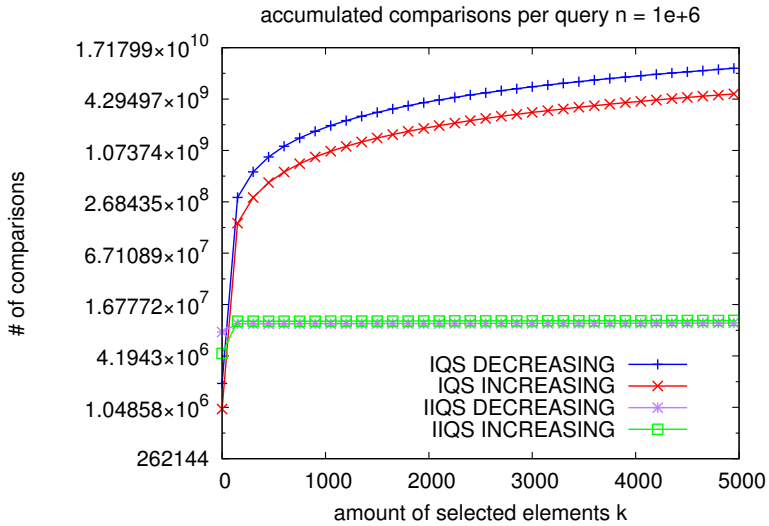
- Intel Xeon CPU E5-2620
- 92GB of RAM
- Experiments were performed for integer arrays with size between 10^2 and 10^6.
- Comparison between algorithms for 10^7 onwards were not possible because of worst-case complexity of IQS.
- Experiments made describe the same behavior between datasets.

Introspective Incremental Quicksort

Experimental results



(a) Key comparisons per each query on IQS and IIQS

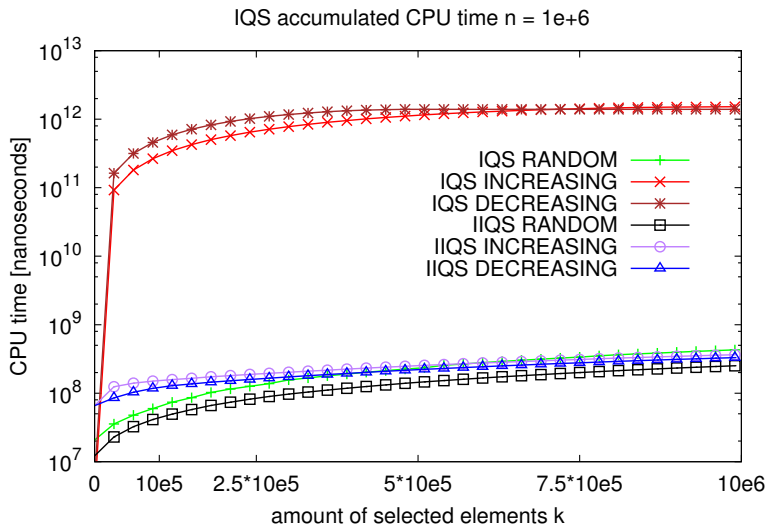


(b) Cumulated key comparisons for IQS and IIQS

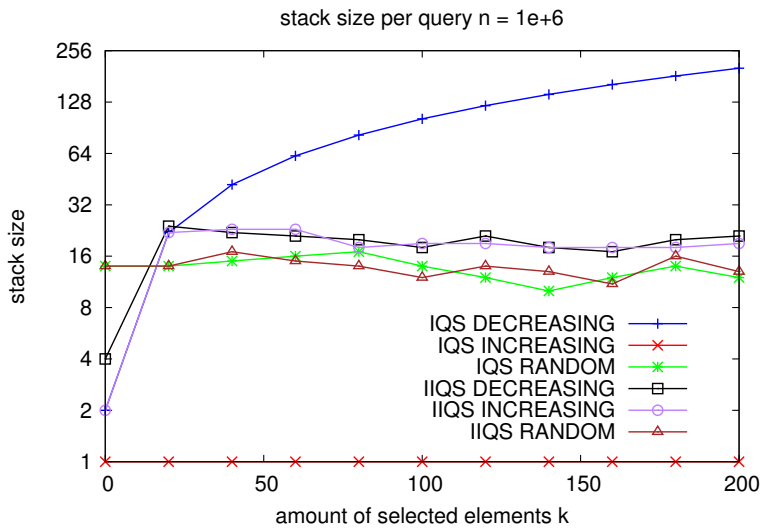
Figure: Key comparisons between IQS and IIQS as a function of the amount of searched elements and per query k. Note the logscales in the plots.

Introspective Incremental Quicksort

Experimental results



(a) CPU time for IQS



(b) Stack size of IQS and IIQS

Figure: Performance comparison between IQS and IIQS as a function of the amount of searched elements k. Note the logscales in the plots.

Conclussions

- We have presented IIQS, an introspective version of IQS to, given a fixed set, retrieve its k smallest elements in optimal time even for the worst case when k is unknown beforehand.
- Pivot selection refinement has a great impact on the performance of both IQS and IIQS.
- IIQS show great competitiveness against the current version of IQS on both worst case and expected case.
- IIQS in practice has the same complexity as IQS plus an overhead on its first call due to the introspective step if this actually occurs and allows the use of a fixed size stack.

Future work

- Analysis for average case for several minima extraction (in progress).
- Compare IIQS with other worst case minima selection algorithm like binary heaps (in progress).

Thank you.

Worst-case Optimal Incremental Sorting

Erik Regla Rodrigo Paredes

Departamento de Ciencias de la Computación
Universidad de Talca

XXXIV International Conference of the
Chilean Computer Science Society (SCCC), 2015