

Relazione Algoritmi e Strutture Dati

Eduard Antonovic Occhipinti, Iman Solaih, Marco Molica

April 29, 2022

Contents

0.1	Quick Sort	2
0.1.1	Impatto della scelta del pivot nel quick sort	2
0.1.2	Fallback a Insertion Sort	8
0.2	Binary Insertion Sort	8
0.3	Skip List	9
0.4	Minimum Heap	22
0.5	Graph	23

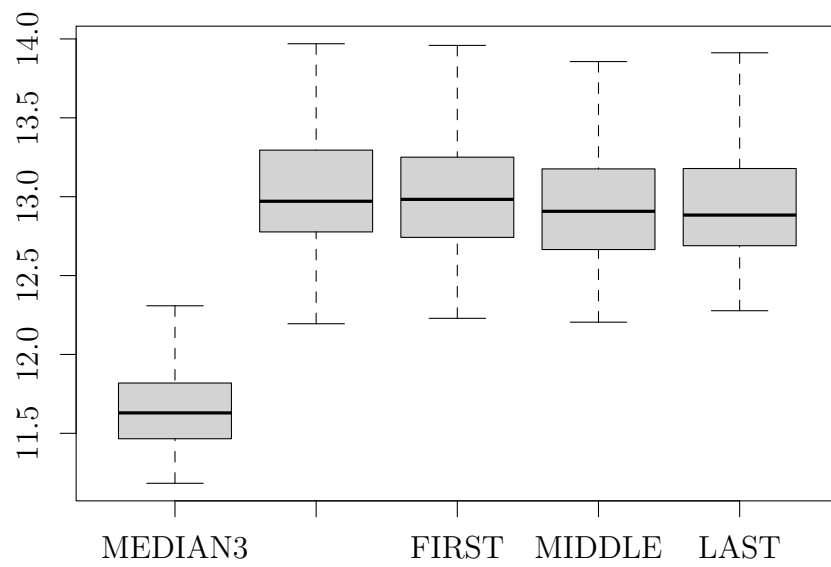
Esercizio 1

0.1 Quick Sort

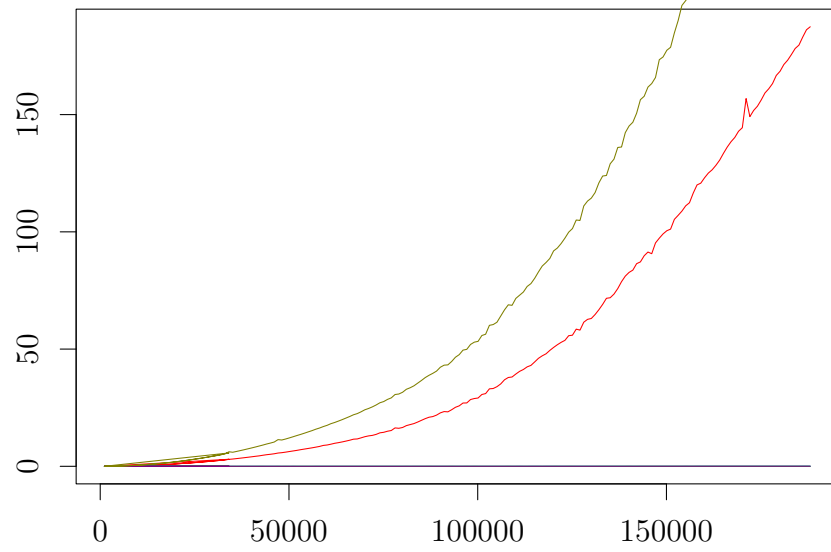
Il `quick_sort()` è un algoritmo che ordina una collezione partendo da un pivot, il pivot può essere scelto in vari modi, e in base a quale viene scelto il tempo di sorting varia. Il `quick_sort()` utilizza `_part()` per scegliere il pivot prima di chiamare `partition()` per dividere gli elementi del range selezionato in un sottoinsieme di elementi maggiori e uno di elementi minori del pivot la cui posizione finale viene restituita dal metodo.

0.1.1 Impatto della scelta del pivot nel quick sort

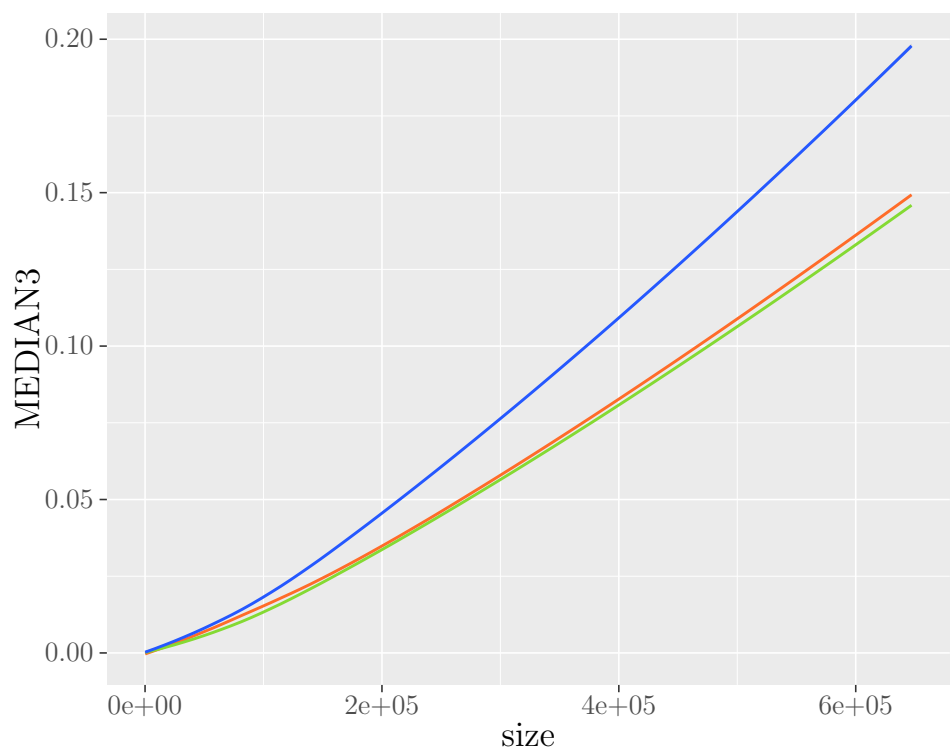
La chiamata a `rand()` porta il `quick_sort()` con pivot scelto randomicamente o come mediana di tre numeri ad essere mediamente più lento rispetto agli altri 3 casi presi in considerazione. La tabella sottostante riporta il tempo impiegato ad ordinare un array di 20 milioni elementi di tipo `struct Record`



La scelta del pivot diventa importante quando l'array in input risulta già parzialmente o totalmente ordinato. Il grafico sottostante riporta il tempo impiegato da `quick_sort()` per scorrere un array già ordinato.



Concentrandoci in particolare sui pivot `median of 3`, `random` e `middle`, possiamo notare che anche tra questi 3 ve ne è uno preferibile rispetto agli altri (aggiungi qualcosa)



0.1.2 Fallback a Insertion Sort

Quando il `quick_sort()` lavora su un range sufficientemente piccolo, è più efficiente utilizzare il `insert_sort()`. Il range di cutoff è stato impostato a 8 elementi.

0.2 Binary Insertion Sort

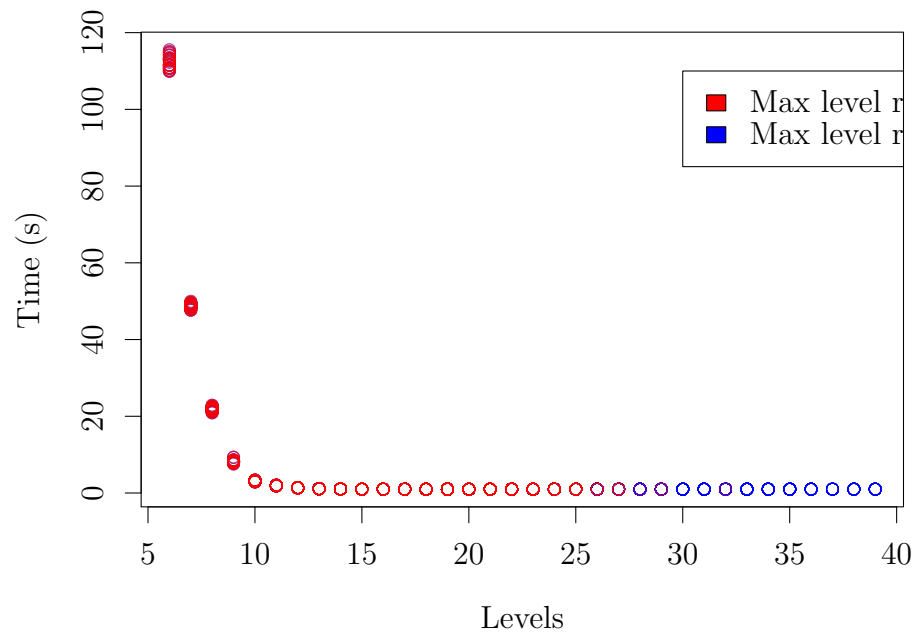
‘ Essendo l’algoritmo di complessità $O(n^2)$, non ci aspettiamo che finisca in tempi sensati l’ordinamento dei 20 milioni di records, facendo due calcoli sui nostri computer dovrebbe metterci approssimativamente 2 anni.

Esercizio 2

0.3 Skip List

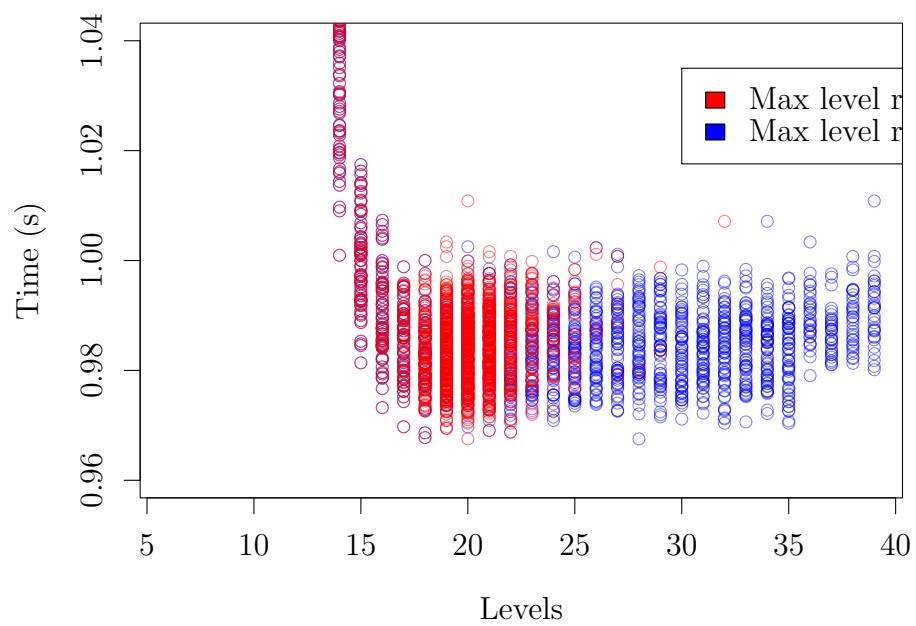
Bla bla bla insertion time decresce in maniera importante

Skip List Insert

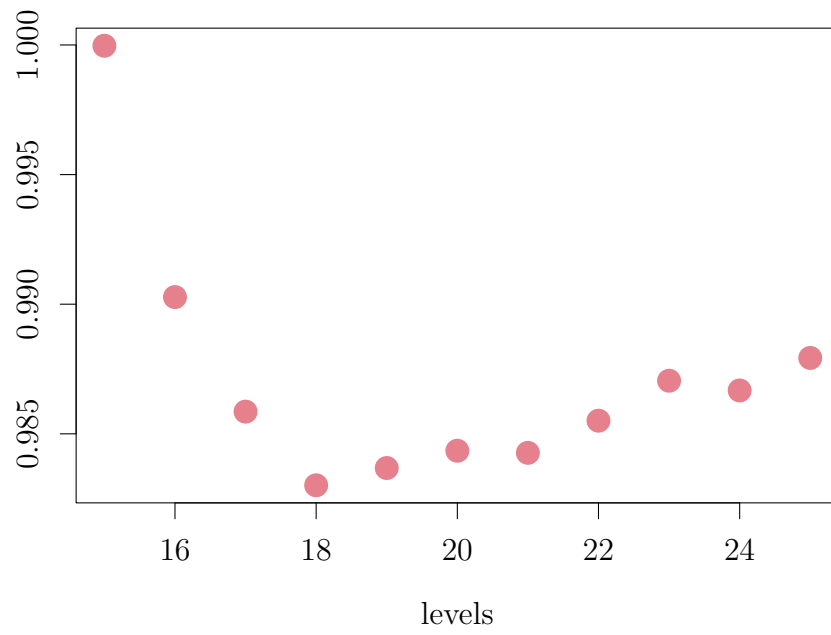


Bla bla bla in particolare zoommando sui livelli più di interesse ci rendiamo conto che la distribuzione è concentrata attorno a 19

Skip List Insert

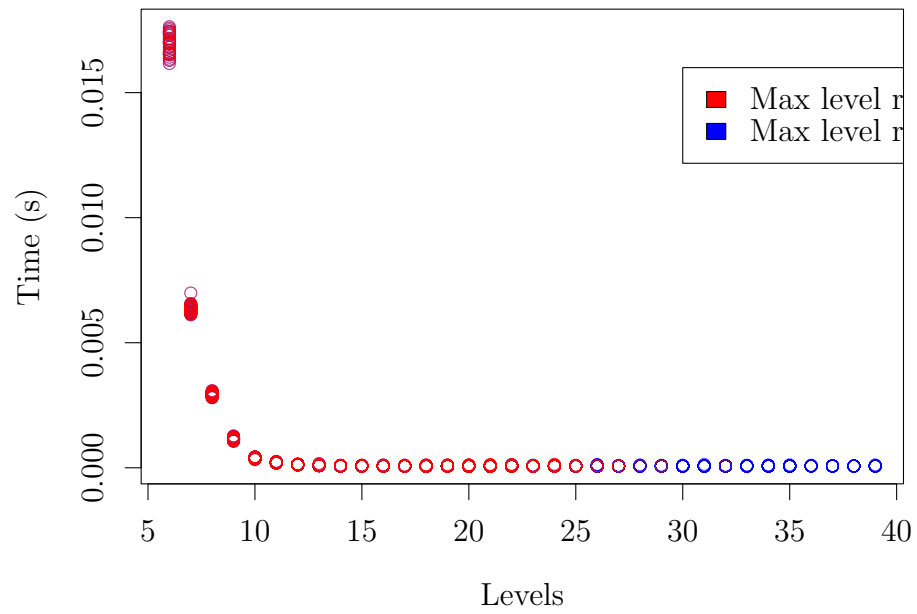


Bla bla bla facendo un grafico delle medie dei tempi di inserimento notiamo che 18 è il numero ottimale di livelli



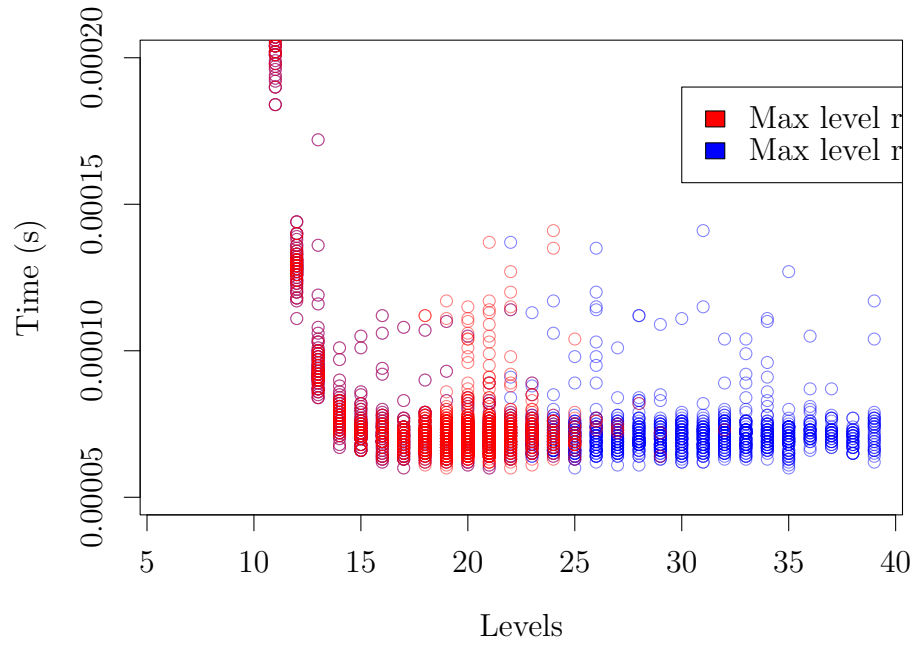
Bla bla bla search time decresce in maniera imporante

Skip List Search

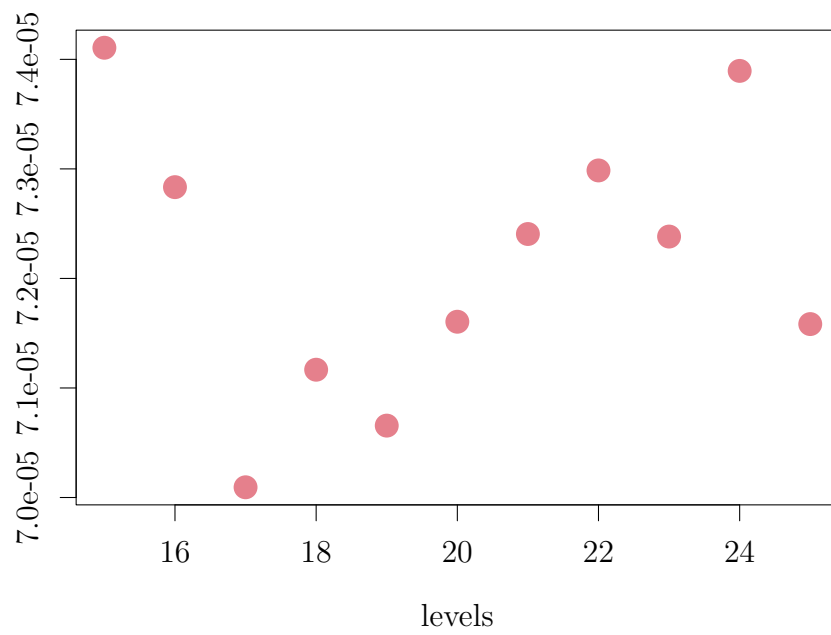


Bla bla bla in particolare zoommando sui livelli più di interesse ci rendiamo conto che la distribuzione è concentrata attorno a 19

Skip List Search



Bla bla bla facendo un grafico delle medie dei tempi di inserimento notiamo che 17 è il numero ottimale di livelli



Sorprendentemente il numero ottimale di livelli non coincide esattamente con $\ln(n)$

Esercizio 3

0.4 Minimum Heap

Esercizio 4

0.5 Graph