Milordo Vincenzo, matricola 913070
Naretto Oscar, matricola 919602
Sisinni Daniele, matricola 914874

Corso B

Algoritmi e Strutture Dati

I° CONSEGNA

Organizzazione del codice:

Il codice è organizzato come da standard nelle cartelle bin, build, src, contenenti rispettivamente i file binari, oggetto e sorgenti. Oltre che l'implementazione del makefile, che permette il passaggio di parametri da linea di comando, è stata sviluppata anche la unit test, con utilizzo di Unity.
Le istruzioni per l'esecuzione sono presenti nel file generic_array_main.c e vengono stampate a schermo in caso di errore nel passaggio dei parametri richiesti.

Algoritmo di riordinamento:

L'algoritmo implementato si basa sulla chiamata ricorsiva dell'algoritmo merge-sort, avente come guardia del caso base il controllo della lunghezza della struttura passata come argomento alla funzione. Se tale struttura ha dimensione minore della costante K (definita di default ma sovrascrivibile da linea di comando), allora si passa ad un approccio iterativo dell'insertion-sort che utilizza la ricerca binaria per migliorare la complessità asintotica.

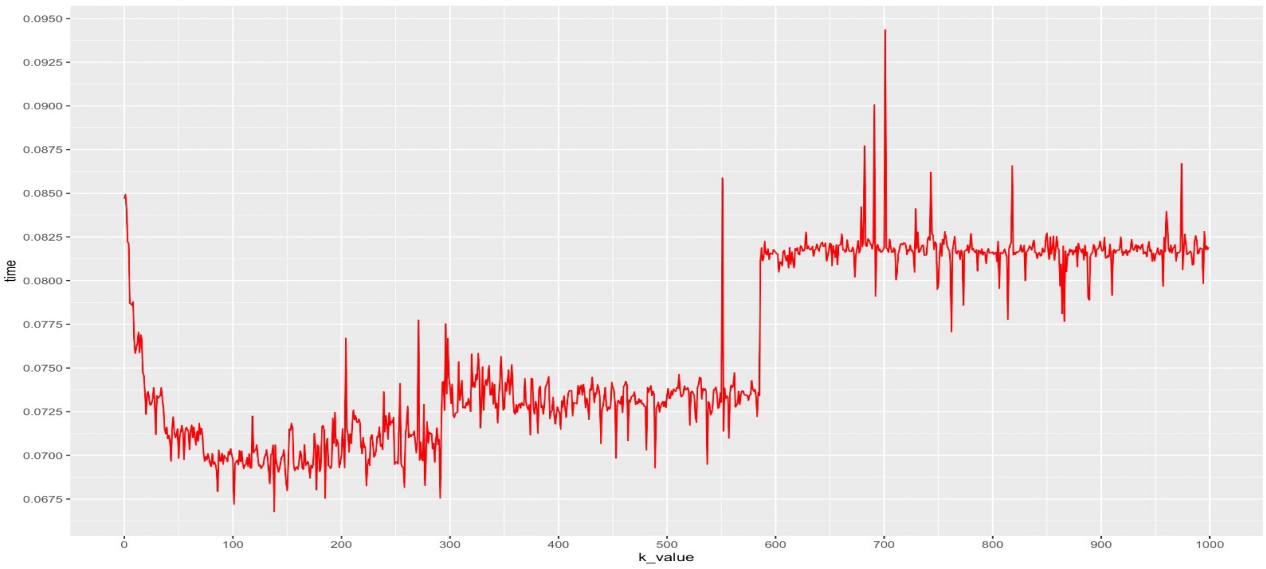
L'implementazione generica dell'algoritmo e della libreria si basa sull'utilizzo dei metodi di confronto precedes_ e succedes_, dando la possibilità di scegliere un ordinamento crescente o decrescente per ciascun campo. Queste due funzioni prendono come argomento due puntatori a void, sfruttando quindi il passaggio di tipi opachi e rendendo generica la libreria.

Studio della costante K:

Per comprendere meglio l'incidenza di questa costante, sono stati effettuati test automatizzati* sul comportamento dell'algoritmo ibrido, prelevandone i tempi di ordinamento in funzione del valore di K, da 0 a 1000.

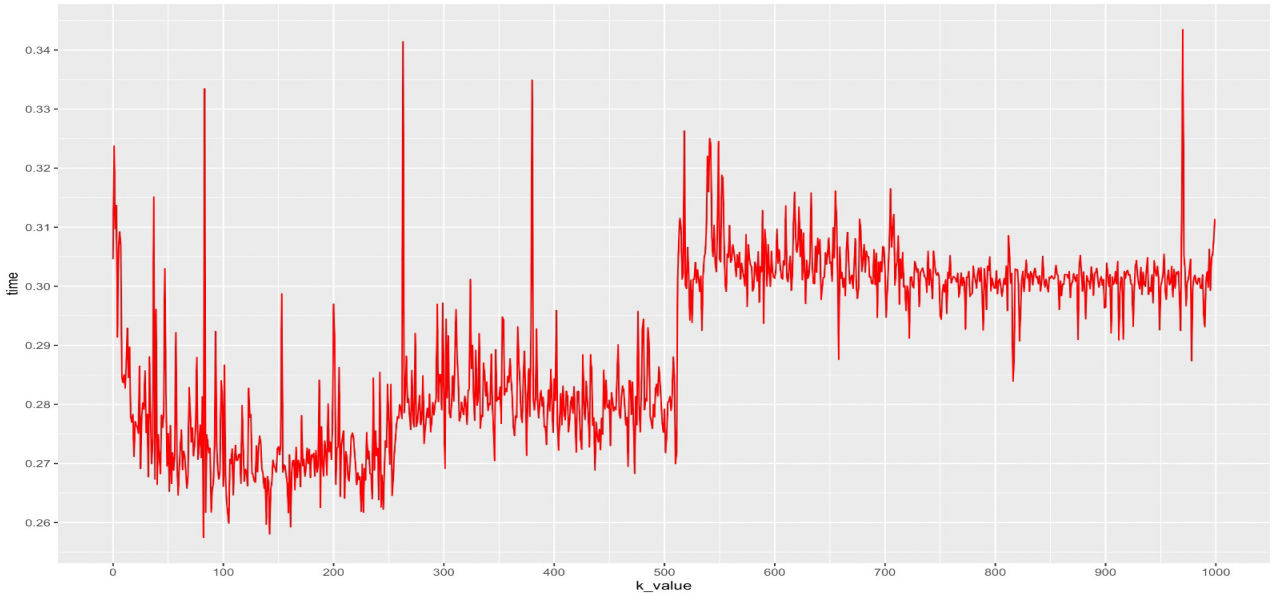
Qui di seguito alcuni grafici** dei tempi rilevati in funzione del K, con strutture di lunghezza variabile.

Struttura di lunghezza 30.000

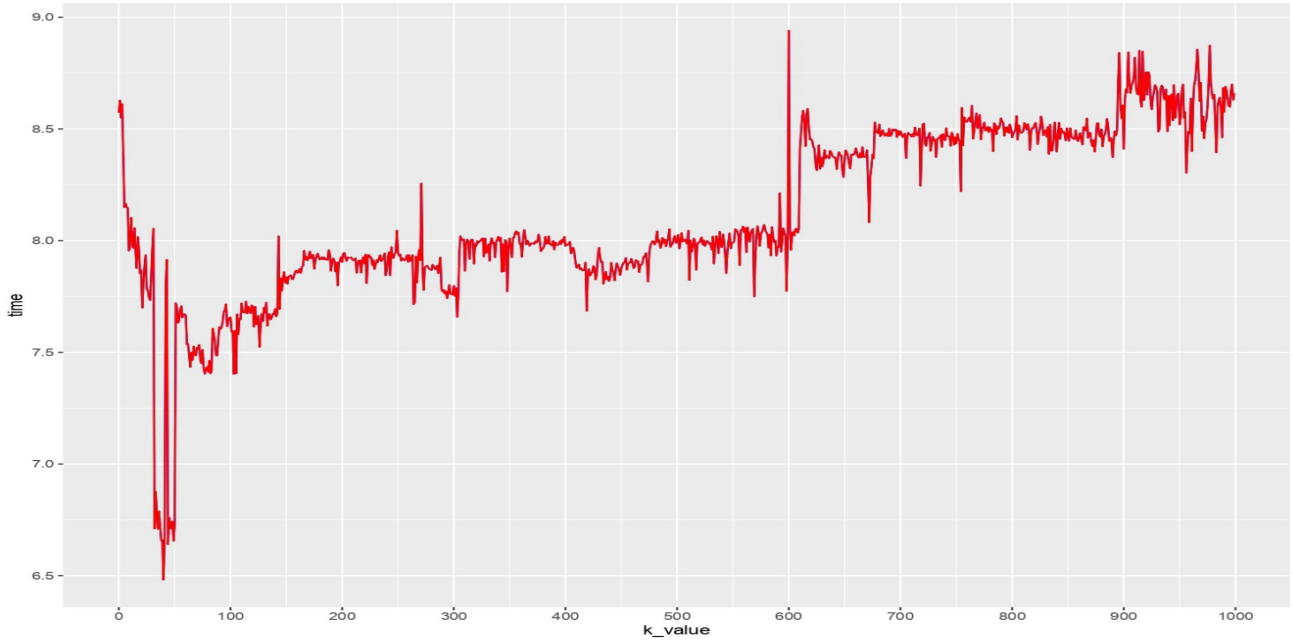


Milordo Vincenzo, matricola 913070
Naretto Oscar, matricola 919602
Sisinni Daniele, matricola 914874

Struttura di lunghezza 1.000.000



Struttura di lunghezza 20.000.000



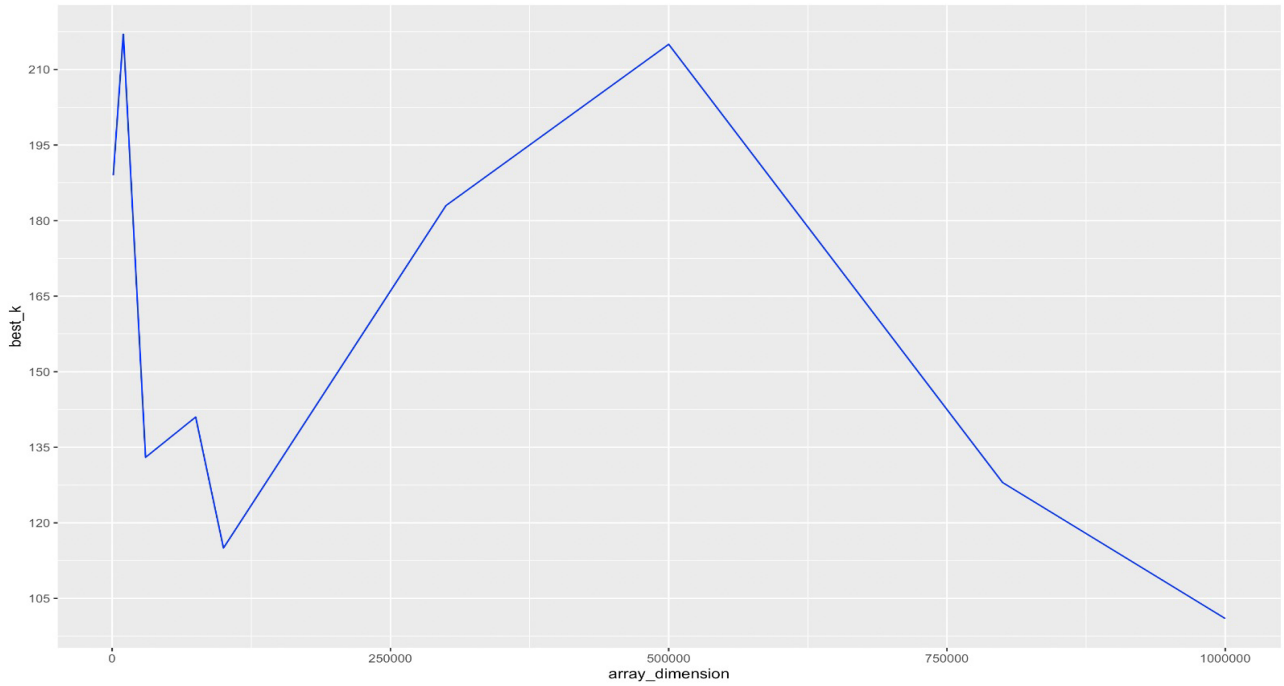
Innanzitutto possiamo osservare, com'era previsto, che per valori molto grandi di K l'efficienza dell'algoritmo diminuisce drasticamente. Questo è comprensibile, poiché il BinaryInsertion-sort si comporta meglio con strutture di piccole dimensioni. Nell'esecuzione dell'algoritmo sul record completo da 20.000.000 infatti si nota un picco di minimo del tempo, con K compreso tra 25 e 50 circa: possiamo ipotizzare che l'algoritmi di BinaryInsertion-sort e di merge-sort, pur avendo complessità simili su strutture di dimensioni K, debbano la loro differenza di prestazioni alla gestione dello spazio alle strutture dati.

Il merge sort infatti, pur essendo un algoritmo ottimo, richiede la generazione di una ulteriore struttura per immagazzinare i valori riordinati e "riuniti" con il merge. Il BinaryInsertion-sort invece, pur avendo complessità asintotica maggiore nel caso generico, su strutture di piccole dimensioni risulta più performante, limitandosi a trovare la posizione, tramite ricerca binaria, dell'elemento corrente.

Da evidenziare che i dati dei tempi descritti dai grafici presentano un notevole margine di variazione da esecuzione ad esecuzione con la macchina*** utilizzata: in particolare con tempi minori ad 1'' abbiamo registrato variazioni addirittura maggiori del 20%, determinata da fattori esterni all'esecuzione del programma stesso.

Notiamo inoltre che per i valori di K nell'intorno di 300 e 600, in tutti e tre le strutture di diversa lunghezza analizzate, si registra un salto nei tempi di ordinamento dei dati.

Come ultima osservazione, presentiamo di seguito un grafico* che mostra la variazione della costante K con il minor tempo di esecuzione (best_k) in funzione della dimensione della struttura da ordinare.



Come già chiarito in precedenza, con strutture minori di 50.000 è presente un margine di errore molto alto; la significatività del grafico soprastante è perciò marginale. Si può però notare che il best_k della struttura di lunghezza 1.000.000 si muova verso un valore molto lontano dagli altri registrati. Questo valore è inoltre coerente con il **best_k del record completo di 20.000.000, avente come valore 35**, che quindi accenna ad una stabilizzazione del K ottimale su valori minori di 100 per strutture di grande dimensioni.

*I grafici sono stati ottenuti attraverso l'elaborazione dei file .csv** con RStudio

I .csv usati come base per i grafici sono stati ottenuti tramite la stampa su file durante l'esecuzione di una versione del programma consegnato, modificata appositamente per limitarsi alla stampa su file dei valori richiesti. Le analisi sono state effettuate su valori manipolabili in tempi accettabili. Tutte le misurazioni sono state fatte su ordinamento **crescente per il campo degli **interi**.

***Macchina di esecuzione: MacBook Air 2020; Sistema Operativo: MacOS 11; RAM: 16 GB; processore: Apple Silicon M1.