

Práctica 02

DOCENTE	CARRERA	CURSO
Vicente Machaca Arceda	Maestría en Ciencia de la Computación	Algoritmos y Estructura de Datos

PRÁCTICA	TEMA	DURACIÓN
02	–	–

1. Datos de integrantes

- Grupo: 2
- Integrantes:
 - EDER ALONSO AMPUERO ATAMARI
 - HOWARD FERNANDO ARANZAMENDI MORALES
 - JOSE EDISON PEREZ MAMANI
 - HENRRY IVAN ARIAS MAMANI

2. Url GITHUB

Repositorio Github: Práctica 2

3. Algoritmo de Ordenamiento

3.1. BTree

El Merge Sort es un algoritmo recursivo bastante eficiente para ordenar un array, que tiene un orden de complejidad $O(n \log n)$ al igual que Quick Sort. fue desarrollado en 1945 por John Von Neumann.

El Merge Sort está basado en la técnica de diseño de algoritmos Divide y Vencerás, esta técnica consiste en dividir el problema a resolver en sub problemas del mismo tipo que a su vez se dividirán, mientras no sean suficientemente pequeños o triviales.

- Si S tiene uno o ningún elemento, está ordenada.
- Si S tiene al menos dos elementos se divide en dos secuencias S1 y S2.
- S1 contiene los primeros $n/2$ elementos y S2 los restantes.
- Ordenar S1 y S2, aplicando recursivamente este procedimiento
- Mezclar S1 y S2 en S, de forma que ya S1 y S2 estén ordenados
- Veamos ahora como sería la estrategia para mezclar las secuencias:

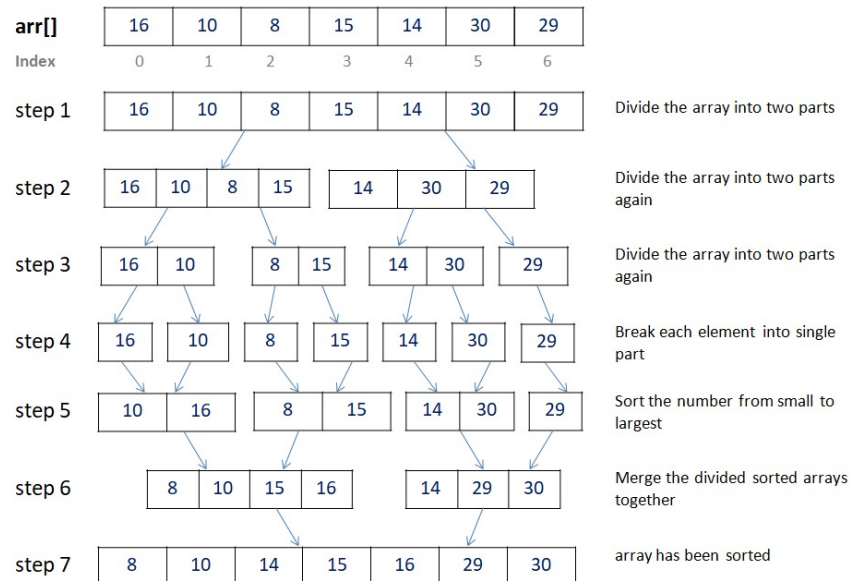


Figura 1: Estrategia que sigue algoritmo para ordenar una secuencia S de n elementos

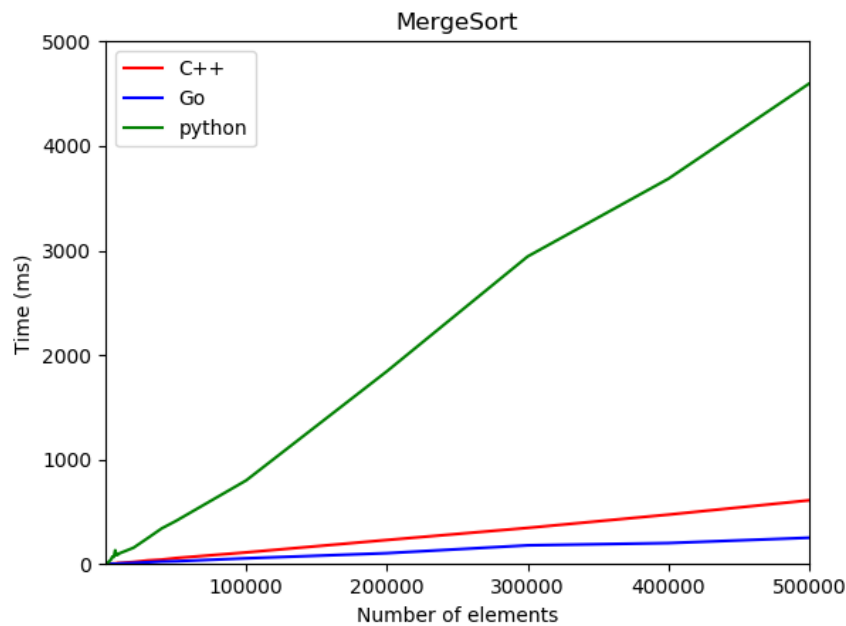


Figura 2: Estrategia que sigue algoritmo para ordenar una secuencia S de n elementos

Se tienen referencias al principio de cada una de las secuencias a mezclar (S1 y S2). Mientras en alguna secuencia queden elementos, se inserta en la secuencia resultante (S) el menor de los elementos referenciados y se avanza esa referencia una posición.

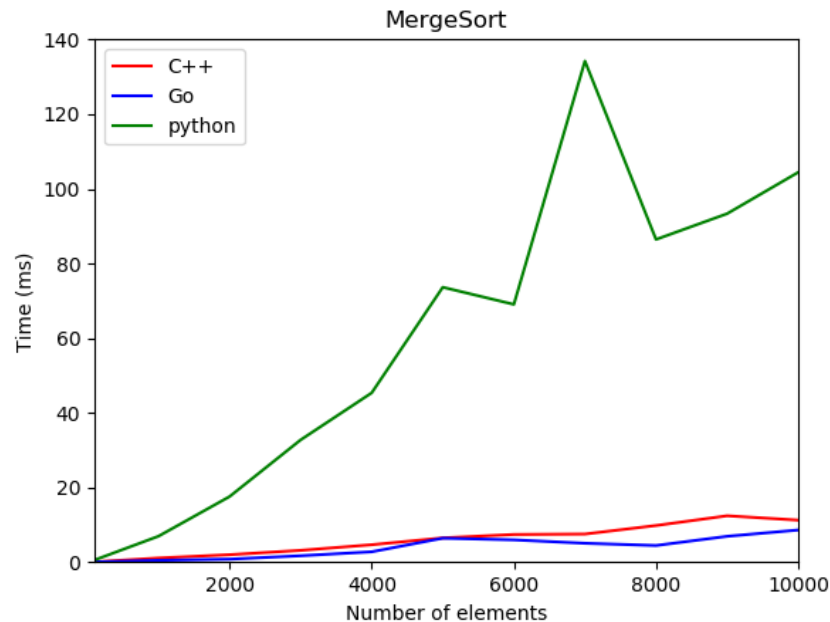


Figura 3: Estrategia que sigue algoritmo para ordenar una secuencia S de n elementos

3.2. AVL

La clasificación de árbol es un algoritmo de clasificación que se basa en la estructura de datos del árbol de búsqueda binaria. Primero crea un árbol de búsqueda binario a partir de los elementos de la lista o matriz de entrada y luego realiza un recorrido en orden en el árbol de búsqueda binario creado para ordenar los elementos.

3.2.1. Costo Computacional

3.2.2. Resultado de las pruebas

4. Conclusiones

- En las pruebas realizadas para el Algoritmo Quick Sort, se obtuvo tiempos de ejecución menores para el código desarrollado en lenguaje de programación Golang, y tiempos de mayor valor en la ejecución del código en lenguaje de programación Python.
- Se observó que, para tamaños de entrada menores a 10 000 datos, los tiempos de ejecución son inconsistentes, al ejecutar el código del algoritmo Quick Sort en lenguaje Golang, presentándose en reiteradas oportunidades valores de cero.
- En la ejecución del algoritmo Quick Sort, los valores de desviación estándar son mayores al ejecutar el código en lenguaje Python, y presentan valores menores al usar el código en C++. Para las pruebas realizadas en Python se observa que los valores de desviación estándar van en aumento respecto al tamaño de la entrada, en el caso del lenguaje Golang y C++, estos valores se incrementan desde 100 000 y 20 000 datos, respectivamente.
- Los programas en lenguaje compilado (C++) presenta un mejor desempeño en cuanto se presenta mayor cantidad de datos que un programa en lenguaje interpretado (Python)

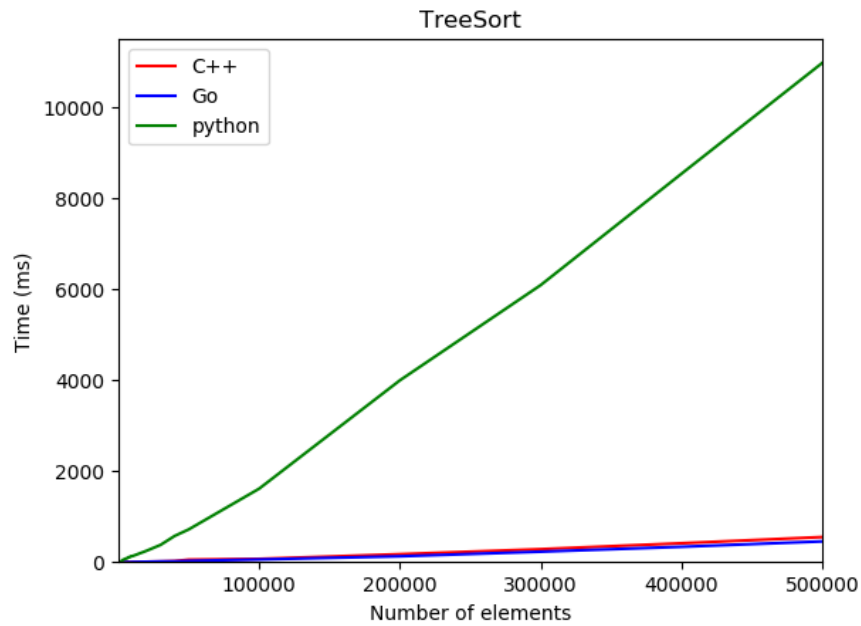


Figura 4: Estrategia que sigue algoritmo para ordenar una secuencia S de n elementos

5. Referencias

Algoritmo: Tree Sort							
				Lenguaje: C++			
N de Datos	t1	t2	t3	t4	t5	Promedio(t)	desv. s.
100	0.0345	0.0215	0.0282	0.0198	0.0248	0.02576	0.005850897
1000	0.2949	0.2878	0.2575	0.2509	0.3067	0.27956	0.024227216
2000	0.5418	0.6543	0.4691	0.8348	0.4672	0.59344	0.154937287
3000	1.2841	1.3463	0.7256	0.761	1.2948	1.08236	0.310662217
4000	1.4992	0.9804	1.2329	2.0372	1.1113	1.3722	0.418131122
5000	3.241	1.7043	1.6303	2.3498	1.629	2.11088	0.70048675
6000	2.3611	3.3129	4.3816	2.7473	3.162	3.19298	0.761382471
7000	2.6891	3.2629	2.348	2.3321	3.9669	2.9198	0.69636647
8000	7.2664	3.7294	8.6777	4.3366	5.2377	5.84956	2.071491642
9000	3.2077	3.1763	4.7729	3.4111	3.5677	3.62714	0.659954353
10000	3.5995	3.7944	3.8475	3.9555	7.1328	4.46594	1.496399306
20000	8.9341	12.2247	22.0834	9.514	9.0016	12.35156	5.605305854
30000	20.4284	15.7075	14.5408	16.3092	15.609	16.51898	2.276363981
40000	24.0826	23.9909	33.5676	22.5173	26.759	26.18348	4.402227591
50000	122.234	42.6545	44.3492	37.6127	39.9763	57.36534	36.35353237
100000	70.2404	77.7421	78.2011	70.5764	77.8051	74.91302	4.117612814
200000	174.7876	193.7067	172.2818	172.4993	180.8891	178.8329	9.011849659
300000	286.8869	287.1223	310.3984	279.5616	280.2003	288.8339	12.57241663
400000	404.6286	437.1003	426.6823	414.3129	413.0981	419.16444	12.74599799
500000	545.8179	552.81	560.67	551.5422	565.9059	555.3492	7.930026684

Tabla 1: Tabla de resultados Tree Sort con C++

Algoritmo: Tree Sort							
				Lenguaje: GO			
N de Datos	t1	t2	t3	t4	t5	Promedio(t)	desv. s.
100	0	0	0	0	0	0	0
1000	0	0	0	0.5229	0	0.10458	0.233847989
2000	1.0602	0	0.5173	0.5213	0.6305	0.54586	0.377853004
3000	0.5163	0.5263	0.5168	1.0277	0.4843	0.61428	0.231653733
4000	1.4224	2.1089	1.9989	1.9995	1.5577	1.81748	0.305999987
5000	1.5459	1.4908	0.9963	1.7008	1.8522	1.5172	0.323570479
6000	2.5209	2.6206	2.9977	2.2495	1.5574	2.38922	0.536789164
7000	3.6788	9.0697	4 2.9965	1.6009	4.26918	4.26918	2.837491464
8000	2.0718	2.0822	4.7402	5.9971	6.4824	4.27474	2.104585229
9000	2.0692	2.8883	6.0346	4.5788	4.0877	3.93172	1.534833267
10000	8.5031	7.0278	7.0258	8.9303	3.0253	6.90246	2.33117331
20000	16.9883	9.9934	11.4141	11.2215	10.4236	12.00818	2.843572749
30000	20.9716	21.9881	16.9907	19.2206	15.989	19.032	2.547690131
40000	17.9895	21.9853	20.9874	21.9866	22.9862	21.187	1.922082861
50000	33.9807	29.9834	28.9829	31.979	26.983	30.3818	2.700491356
100000	60.0404	61.9663	63.9625	53.97	72.955	62.57884	6.901271991
200000	116.2649	130.0599	145.3944	140.5367	129.0323	132.25764	11.31501115
300000	214.4081	218.5863	282.0565	201.3208	253.0039	233.87512	33.01458859
400000	305.4279	307.92	367.4944	341.7326	384.1765	341.35028	35.09183418
500000	419.8142	429.1118	493.0747	515.5093	442.0059	459.90318	42.03551579

Tabla 2: Tabla de resultados Tree Sort con GO

Algoritmo: Tree Sort							
				Lenguaje: Python			
N de Datos	t1	t2	t3	t4	t5	Promedio(t)	desv. s.
100	0.015624285	0	0.001001835	0.000999928	0	3.525209427	6.782077381
1000	0.008558035	0.003997564	0.004995108	0.004993439	0.010315895	6.572008133	2.718788759
2000	0.019988537	0.026985168	0.028544664	0.011993408	0.021991014	21.90055847	6.553873617
3000	0.024986029	0.042976379	0.049971104	0.049968719	0.079953671	49.57118034	19.82003258
4000	0.040974855	0.091937304	0.045972109	0.070964336	0.067958117	63.56134415	20.60862468
5000	0.044970989	0.11288166	0.042973995	0.101940393	0.09394598	79.34260368	32.98814407
6000	0.070955515	0.071353912	0.058964968	0.099941492	0.108936548	82.03048706	21.29203294
7000	0.091950655	0.099225044	0.135372639	0.132360697	0.10193944	112.1696949	20.16856982
8000	0.120291233	0.094572067	0.080644131	0.139489412	0.112987995	109.5969677	22.82235213
9000	0.113935947	0.173901558	0.073012352	0.108042202	0.212877989	136.3540096	56.08482712
10000	0.132925034	0.090055466	0.129489422	0.1939466	0.133127928	135.9088898	37.17781267
20000	0.315265894	0.239135265	0.208152771	0.246989012	0.255201578	252.948904	39.12004677
30000	0.378284216	0.381990671	0.358812094	0.447550774	0.349073887	383.1423283	38.49024037
40000	0.55368185	0.62367034	0.560783863	0.627627611	0.533614874	579.8757076	42.97934242
50000	0.716868401	0.73401022	0.728899479	0.708563566	0.729392529	723.5468388	10.50482242
100000	1.756626368	1.585929394	1.614357233	1.669086695	1.468574524	1618.914843	106.292876
200000	3.741744757	3.7024014	3.825145483	3.708697319	5.033451557	4002.288103	578.508426
300000	6.054645538	5.865937948	6.147264957	5.924173594	6.492034435	6096.811295	246.7950222
400000	8.224924326	8.243051291	8.770008802	8.109434605	9.380488396	8545.581484	531.9790418
500000	9.972488165	11.47141552	10.28874731	10.39334488	12.78968287	10983.13575	1156.877989