КВАНТИТАТИВНА ОЦЕНА ЕФИКАСНОСТИ АЛГОРИТАМА СОРТИРАЊА У ПРОГРАМСКИМ ЈЕЗИЦИМА C#, JAVA И PYTHON

Милош Илић [1] Лазар Копања [2,3] Александр Михайлович Недзьведь [4,5] Драган Златковић [6]

Резиме: У рачунарским наукама као мерило перформанси и ефикасности система понајвише се користе алгоритми сортирања. Сваки алгоритам сортирања као и сваки програмски језик има одређене предности и недостатке. Са циљем осветљавања те чињенице, овај рад се бави перформансама извршавања одабраних алгоритама сортирања који се користе у различитим модерним рачунарским језицима/компајлерима (C#, Java и Python). Алгоритми сортирања тестирани су за насумично генерисане низове података различите величине и структуре. Резултати показују интересантне разлике у ефикасности сортирања између појединих имплементација у односу на оптерећење CPU-а и време извршења. Поред емпиријског дискутован је и аналитички начин одређивања времена извршавања изабраних алгоритама.

Кључне речи: Алгоритми сортирања, перформансе алгоритама, C#, Java, Python

QUANTITATIVE EVALUATION OF SORTING ALGORITHM EFFICIENCY IN C#, JAVA AND PYTHON PROGRAMMING LANGUAGES

Abstract: In computer sciences, sorting algorithms are used as the prevalent benchmark of system perfomance and efficiency. Every sorting algorithm, just like every programming language, has its advantages and disadvantages. In order to shed further light on this matter, this paper deals with the execution performance of chosen sorting algorithms used in various modern programming languages/compilers (C#, Java and Python). Sorting algorithms were tested on randomly generated arrays of varying size and structure. The results indicate noteworthy differences in sorting efficiency among individual implementations in relation to CPU load and execution time. Also discussed in this paper, apart from the empirical, is the analytical method for determining the execution time of the chosen algorithms.

Keyword: Sorting algorithms, algorithm performance, C#, Java, Python

1. УВОД

Алгоритми сортирања се претходних деценија темељно изучавају у литератури (Munro и сар., 1996; Blelloch и сар., 1998; Smith, 2004; Gerbessiotis, 2012). Alnihoud и сар. (2010) оцењују сортирање као један од основних проблема у рачунарским наукама, с обзиром на то да највећи број апликација има потребу за неком врстом сортирања података.

Многи аутори представљају своја решења у повећању ефикасности стандардних алгоритама сортирања. Sfrancis и сар. (1992) излажу начин на који се може унапредити Quicksort. Zhang и сар. (1996) описују како убрзати Merge sort, док Saher и сар. (2017) представљају двосмерни условни Insertion sort који је много ефикаснији од стандардног. Michael Codish и сар. (2017) предлажу оптимизацију алгоритама сортирања коришћењем мрежа за сортирање. Неки аутори побољшање перформанси

[1] Факултет информационих технологија, Алфа БК Универзитет, Београд, Србија, milos.ilic@alfa.edu.rs:
[2] Факултет информационих технологија, Алфа БК Универзитет, Београд, Србија, lazar.kopanja@alfa.edu.rs:
[3] Технолошко-металуршки факултет, Универзитет у Београду, Београд, Србија lazar.kopanja@alfa.edu.rs:
[4] Белоруски државни универзитет, Минск, Белорусија, nedzveda@tut.by:
[5] Институт за информатичке проблеме Националне академије наука Белорусије, Минск, Белорусија:
[6] Факултет информационих технологија, Алфа БК Универзитет, Београд, Србија, dragan.zlatkovic@alfa.edu.rs:

алгоритама сортирања виде у њиховим комбинацијама, односно стварању такозваних „хибридних“ алгоритама за сортирање. Chowdhury и сар. (2000) представљају Heap-Mergesort algoritam. Diekert и сар. (2016) анализирају QuickHeapsort алгоритам, док Edelkamp и сар. (2019) предлажу занимљиво решење QuickXsort који је комбинација Quicksort-а са неким од стандардних алгоритама сортирања као што су Merge sort, Heap sort или Insertion sort (X у називу QuickXsort-а значи Merge, Heap или Insertion).

Избор програмског језика приликом креирања апликација може бити од велике важности. Једноставност, флексибилност и робусност су значајне особине које одређени програмски језик квалификују за погодан избор. Поред тога, фактори који се узимају у обзир приликом одабира програмског језика су продуктивност програмера, одрживост, преносивост, подршка за алате и ефикасност. Одабир програмског језика представља проналажење финог баланса наведених параметара. Брзина извршавања је једна од најважнијих карактеристика апликације. Стога се увек тежи да у смислу брзине извршавања перформансе буду што боље што, наравно, веома зависи и од програмског језика који се користи при развоју апликације. Колико је ефикасност програмских језика важна показује и велики број научних радова у којима се пореде различити програмски језици ради проналажења најефикаснијег могућег решења. Moreira (1989) упоређује програмске језике Java, C/C++ и FORTRAN за потребе нумеричког рачунања. L. Prechelt (2000) је извршио компарацију седам програмских језика C, C++, Java, Perl, Python, Rexx и Tcl како би оценио њихову ефикасност. Fourment и сар. (2008) пореде уобичајене програмске језике који се користе у биоинформатици C, C++, C#, Java, Perl и Python. Stein и сар. (2013) раде компарацију пет програмских језика међу којима су C++, Java, C#, F# and Python.

У овом раду су разматрана три алгоритма сортирања Quicksort, Merge sort i Heap sort кроз три различита програмска језика C#, Java и Python са циљем поређења брзине извршавања ових алгоритама.

2. МАТЕРИЈАЛ И МЕТОД

2.1. Алгоритми сортирања - Quicksort, Merge sort и Heap sort

Quicksort је један од најчешће коришћених алгоритама сортирања који припада групи метода замене (Fouz, 2012). Он функционише тако што се прво изабере један пивот елемент, а затим се елементи који су мањи или једнаки од пивот елемента распоређују са његове леве стране, а они већи десно од њега. Овај процес се рекурзивно понавља на нове партиције све док се не заврши сортирање.

Merge sort такође припада групи метода замене. Алгоритам ради по принципу „подели па владај“ (Nardelli и сар., 2006). Низ који је потребно сортирати се прво подели на два подниза, потом се та два подниза рекурзивно сортирају, да би се на крају та два соритрана дела спојила.

Heap sort подразумева да се прво се креира гомила (heap) од елемената низа. Гомила се потом претвара у максималну гомилу, затим се замени коренски са задњим чвором и на крају се последњи чвор избрише из хрпе. Овај процес се потом понавља све док низ не буде сортиран (Lutz и сар., 1989).

2.2. Окружење и хардвер

Алгоритми су тестирани на рачунару са Windows 10 Pro 64-битним оперативним системом. Процесор рачунара је Intel i7-9700K који има 8 језгара брзине 3.6 GHz (4.9 GHz Turbo) и 12MB кеш меморије (L1 - 512KB, L2 - 2MB i L3 - 12MB). SSD диск је

капацитета 512 GB и има брзину читања 3500 MB/s и писања 2300 MB/s, док је RAM меморија капацитета 32 GB (DDR4).

Испитивање алгоритама у програмском језику C# су рађена у Visual Studio-у 2017. Visual Studio представља интегрисано развојно окружење развијено од стране Microsoft-а које може да се користи за уређивање, дебаговање и изградњу кода.

Када је у питању Java коришћен је NetBeans 8.2 интегрисано развојно окружење које омогућава рад са најновијим Java 8 технологијама (Nosál и сар., 2017). Ова платформа подржава развој свих типова Java апликација.

На крају за програмски језик Python је коришћен IDLE (Python 3.7) интегрисано окружење за развој и учење. Веома је погодан за почетнике и доста се користи у образовном систему.

2.3. Методологија

Време извршавања алгоритма може се одредити емпиријски и аналитички. Аналитичко одређивање времена извршавања заснива се на теоријској анализи алгоритма и пребројавању јединичних инструкција, док емпиријски метод подразумева мерење стварног времена извршавања програма на рачунару. Како је аналитичко време извршавања посматраних алгоритама сортирања добро познато у теорији, овај рад се бави емпиријским временом извршавања на описаном рачунарском систему. У сврху тестирања алгоритама сортирања креирано је осам .txt фајлова који садрже хиљаду, 10 хиљада, 100 хиљада, милион, 2 милиона, 3 милиона, 4 милиона и 5 милиона насумичних целих бројева из интервала [1, 1000]. На овај начин обезбеђено је да алгоритми сортирања при тестирању увек користе исте низове како би резултати били што прецизнији.

Quicksort, Merge sort и Heap sort су потом имплеменирани у три различита програмска језика C#, Java и Python у поменутим окружењима. Сваки алгоритам прво учитава насумичне бројеве из .txt фајла и потом их сортира. Треба напоменути да је време извршавања мерено само за процес сортирања, а не и за учитавање бројева у низ и њихово исписивање након сортирања.

Обзиром да време извршавања истог кода може да варира приликом сваког тестирања, алгоритми су тестирани више пута, а затим је узето просечно време извршавања. Исти поступак је поновљен за посматрана три алгоритма сортирања у сваком од ова три програмска језика.

3. РЕЗУЛТАТИ И ДИСКУСИЈА

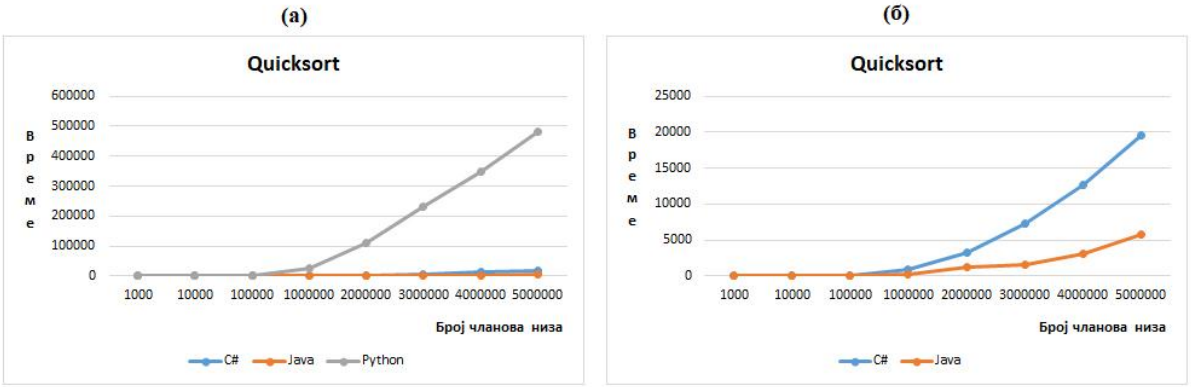
Резултати за време извршавања Quicksort, Merge sort и Heap sort алгоритама за соритрање су приказани у Табели 1. Време је приказано у милисекундама.

На основу резултата из Табеле 1 може се уочити да је време извршавања за сва три алгоритма сортирања најкраће у програмском језику Java. Следећи програмски језик је C#, док је убедљиво најспорији програмски језик Python. Као што је и очекивано са повећањем броја чланова низа расте и време извршавања посматраних алгоритама код сва три програмска језика.

Табела 1 – Време извршавања алгоритама

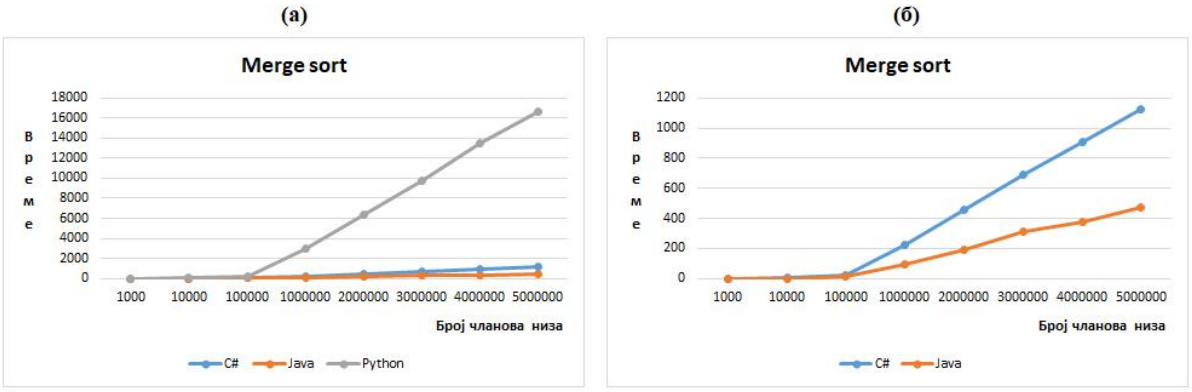
CPU									
Број чланова низа	Quicksort			Merge sort			Heap sort		
	C#	Java	Python	C#	Java	Python	C#	Java	Python
1000	0,37	0,01	4,54	0,588	0,02	4,54	26,234	0,01	149,45
10000	1,27	0,1	16,44	2,322	0,2	33,138	280,744	15,6	484,56
100000	19,07	1	365,44	24,106	15,8	247,544	2261,182	161,2	4410,54
1000000	902,38	190,8	27289,8	223,234	93,4	2921,2	14647,46	1573,6	44458,56
2000000	3313,86	1195,80	109292,2	454,346	194,2	6328,4	29561,02	3151,8	89645,06
3000000	7363,73	1586,8	230946,2	690,016	315,2	9787	41904,25	4701,8	138202,7
4000000	12694,9	3151,6	346958,4	911,91	378,4	13516,4	58238,13	6180	187991,2
5000000	19581,37	5841,4	480109,6	1128,22	468,8	16595,4	66709,71	7695	237872,2

На Слици 1 су приказана два графикона која представљају време извршавања Quicksort алгоритма у односу на број чланова низа. На графикону (а) приказани су резултати сва три програмска језика, где се може приметити велика разлика код програмског језика Python у односу на C# и програмски језик Java, што постаје све израженије у случају сортирања низова од милион чланова и више. На графикону (а) изгледа као да је за посматрани алгоритам C# само незнатно спорији од програмског језика Java и то тек када је у питању број чланова низа од 4 милиона и 5 милиона. Међутим, њихова разлика се прецизније може видети на графикону (б) где се уочава колико је Java бржа од програмског језика C#.



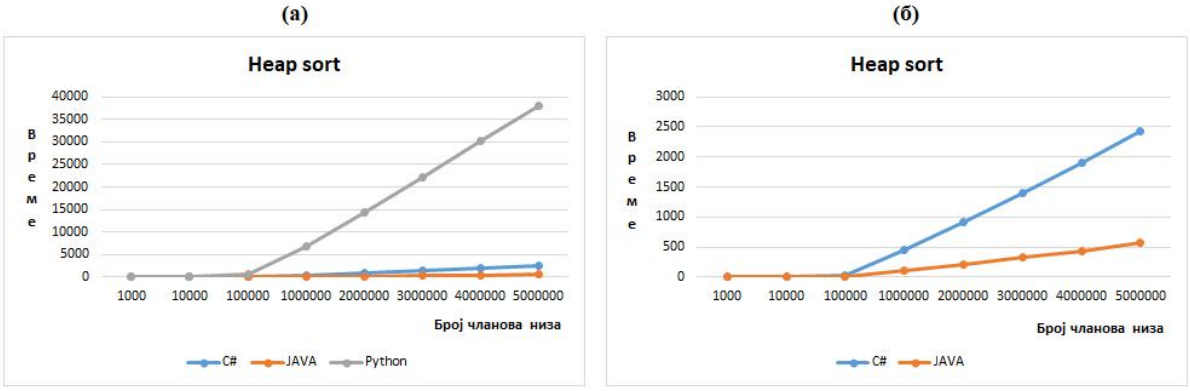
Слика 1 – Време извршавања Quicksort алгоритма у програмским језицима C#, Java и Python

Аналогно претходном алгоритму, Слика 2 приказује два графикона за Merge sort и његово време извршавања. На графикону (а) се могу видети резултати за сва три програмска језика, где као код Quicksort алгоритма постоји велика разлика у времену извршавања када је у питању програмски језик Python у односу на C# и програмски језик Java. Java је бржа у односу на C#, па иако је овде очигледнија разлика, како би се боље упоредила ова два програмска језика њихови резултати су такође издвојени на посебном графикону (б).



Слика 2 – Време извршавања Merge sort алгоритма у програмским језицима C#, Java и Python

Резултати за Heap sort алгоритам се могу видети на Слици 3 где је ситуација иста као и код Quicksort и Merge sort алгоритма у погледу програмских језика. Python је далеко најспорији што се види на графикону (а), док је најбржа Java. На графикону (б) су посебно приказани C# и Java како би се боље сагледала међусобна разлика у времену извршавања.



Слика 3 – Време извршавања Heap sort алгоритма у програмским језицима C#, Java и Python

4. ЗАКЉУЧАК

У овом раду је поређено време извршавања алгоритама сортирања Quicksort, Merge sort и Heap sort кроз три програмска језика C#, Java и Python. Тестирање је извршено у три различита окружења Visual Studio 2017, NetBeans 8.2 и IDLE (Python 3.7), а како би резултати били што веродостојнији за сва тестирања је коришћена иста конфигурација рачунарског система. Сва три алгоритма су анализирана на истим, раније припремљеним, текстуалним фајловима са хиљаду, 10 хиљада, 100 хиљада, милион, 2 милиона, 3 милиона, 4 милиона и 5 милиона насумично изабраних целих бројева из интервала [1,1000]. Узето је у обзир и варирање у времену извршавања које може да се догоди при покретању истог кода, те је сваки алгоритам тестиран више пута да би се потом узело њихово просечно време извршавања. Добијени резултати упућују на

закључак да је под наведеним условима Java најефикаснији програмски језик када је у питању време извршавања анализираних алгоритама сортирања, C# је на другом месту по ефикасности, док Python заостаје у времену извршавања у односу на наведене програмске језике.

5. LITERATURA

[1] Alnihoud, J. & Rami, R. (2010). An Enhancement of Major Sorting Algorithms. *The International Arab Journal of Information Technology*, *7*(1), pp 55-62.

[2] Blelloch G. E., Leiserson C. E., Maggs B. M., Plaxton, C. G., Smith, S. J. & Zagha M. (1998). An Experimental Analysis of Parallel Sorting Algorithms. *Theory of Computing Systems*, *31*(2), pp 135–167. https://doi.org/10.1007/s002240000083

[3] Chowdhury, R. A., Nath, S. K. & Kaykobad, M. (2000). The Heap-Mergesort. *Computers & Mathematics with Applications*, *39* (7–8), pp 193-197. https://doi.org/10.1016/S0898-1221(00)00075-4

[4] Codish, M., Cruz-Filipe, L., Nebeland, M. & Schneider-Kamp, P. (2017). Optimizing sorting algorithms by using sorting networks, *Formal Aspects of Computing*, *29*(3), pp 559–579. https://doi.org/10.1007/s00165-016-0401-3

[5] Diekert, V. & Weiß, A. (2016). QuickHeapsort: Modifications and Improved Analysis. *Theory of Computing Systems*, *59*(2), pp 209–230. https://doi.org/10.1007/s00224-015-9656-y

[6] Edelkamp, S., Weiß, A. & Wild, S. (2019). QuickXsort: A Fast Sorting Scheme in Theory and Practice. *Algorithmica*, *81*, pp 1–80. https://doi.org/10.1007/s00453-019-00634-0

[7] Fourment, M. & Gillings, M.R. (2008). A comparison of common programming languages used in bioinformatics. *BMC Bioinformatics*, *9*(82). doi:10.1186/1471-2105-9-82

[8] Fouz, M., Kufleitner, M., Manthey, B. & Jahromi, N. Z. (2012). On Smoothed Analysis of Quicksort and Hoare's Find. *Algorithmica*, *62*(3–4), pp 879–905. https://doi.org/10.1007/s00453-011-9490-9

[9] Gerbessiotis, A.V. (2012). An improved reliability bound of a probabilistic parallel integer sorting algorithm. *Information Processing Letters*, *112*(24), pp 976-979. https://doi.org/10.1016/j.ipl.2012.09.003

[10] Mohammed, A. S., Amrahov, Ş. E. & Çelebi, F.V. (2017). Bidirectional Conditional Insertion Sort algorithm; An efficient progress on the classical insertion sort. *Future Generation Computer Systems*, *71*, pp 102-112. https://doi.org/10.1016/j.future.2017.01.034

[11] Moreira, J.E., Midkiff, S.P. & Gupta, M. (1989). A comparison of Java, C/C++, and FORTRAN for numerical computing. *IEEE Antennas and Propagation Magazine*, *40*(5), pp 102-105. DOI:10.1109/74.736311

[12] Munro, J. I. & Raman, V. (1996). Fast stable in-place sorting with *O(n)* data moves. *Algorithmica*, *16*(2), pp 151–160. https://doi.org/10.1007/BF01940644

[13] Nardelli, E. & Proietti, G. (2006). Efficient unbalanced merge–sort. *Information Sciences*, *176*(10), pp 1321-1337. https://doi.org/10.1016/j.ins.2005.04.008

[14] Nosál, M., Porubän, J. & Sulír, M. (2017). Customizing host IDE for non-programming users of pure embedded DSLs: A case study, *Computer Languages, Systems & Structures*, 49, pp 101-118. https://doi.org/10.1016/j.cl.2017.04.003

[15] Prechelt, L. (2000). *An empirical comparison of seven programming languages. Computer*, *33*(10), pp 23–9. DOI: 10.1109/2.876288

[16] Sfrancis, R. & Pannan, L.J.H. (1992). A parallel partition for enhanced parallel QuickSort. *Parallel Comput.*, *18*, pp 543–550. https://doi.org/10.1016/0167-8191(92)90089-P

[17] Smith, R. (2004). Comparing Algorithms for Sorting with *t* Stacks in Series. *Annals of Combinatorics*, *8*(1), pp 113–121. https://doi.org/10.1007/s00026-004-0209-3

[18] Stein, M. & Geyer-Schulz, A. (2013). A Comparison of Five Programming Languages in a Graph Clustering Scenario. *Journal of Universal Computer Science*, *19*(3), pp 428-456. DOI:10.3217/jucs-019-03-0428

[19] Zhang, W. & Larson, P. (1996). Speeding up external merge sort. *IEEE Trans. Knowl. Data Eng.*, *8*, pp 322–332. DOI: 10.1109/69.494169

[20] Wegner, L. M. & Teuhola, J. (1989). The external heap sort, *IEEE Transactions on Software Engineering*, *15*, pp 917–925. DOI: 10.1109/32.29490