Table 3: Implementation variants of the selected sorting algorithms. [17]

	Variant 1	Variant 2
Merge Sort	Recursive and without dynamic memory allocation	Recursive and with dynamic memory allocation
Heap Sort	Memory-based swapping	Pointer-based swapping
Quick Sort	Memory-based swapping	Pointer-based swapping
Insertion Sort	Memory-based swapping and no separate function for sorting	Memory-based swapping but additional function to perform sort operation
Bubble Sort	Memory-based swapping	Pointer-based swapping
Selection Sort	Memory-based swapping	Pointer-based swapping

Table 4: Calibrated problem sizes.

	Problem Size (# of integers)		
Algorithm	Small	Medium	Large
Merge Sort	10 240 000 000	10 880 000 000	11 520 000 000
Heap Sort	10 240 000 000	10 880 000 000	11 520 000 000
Quick Sort	10 240 000 000	10 880 000 000	11 520 000 000
Insertion Sort	640 000	12 800 000	19 200 000
Bubble Sort	640 000	12 800 000	19 200 000
Selection Sort	640 000	12 800 000	19 200 000

compare the results, we calculate the performance in *sorted Kb per second.*

4 RESULTS AND ANALYSIS

As the results in Table 5, Table 6, and Table 7 show, Merge Sort, Heap Sort, and Quick Sort are significantly more energy-efficient than Insertion Sort, Bubble Sort, and Selection Sort across all problem sizes and scenarios. This is not surprising, given that the former have an average time complexity of $n * log(n)$ and the latter an average time complexity of n^2.

Within the set of algorithms with time complexity $n * log(n)$, the Merge Sort implementation that uses pre-allocated memory (variant 1) is the most energy-efficient across all problem sizes and in all three scenarios. Its energy efficiency even increases notably with problem size in most scenarios, while for most of the other algorithms, energy efficiency often decreases or only increases by a small margin. As the comparison between Table 5 and Table 6 indicates, the energy efficiency of Quick Sort variant 1 on both servers decreases by a large margin when the data is approximately 20% pre-sorted in comparison to the non-pre-sorted data. For Heap Sort (both implementation variants) on server A, this decrease in energy efficiency also can be observed, but is much less pronounced. On the other hand, for both variants of Merge Sort and Quick Sort variant 2, the energy efficiency increases in this scenario. On server A it can be observed that both variants of Heap Sort are less energy-efficient on the *medium* and *large* problem sizes when the data is 20% pre-sorted. However, Quick Sort variant 2 does not suffer from this effect and is the second most efficient algorithm in these circumstances, after Merge Sort variant 1. Further, it can be seen that Quick Sort variant 1 is more energy-efficient on the non-pre-sorted data than variant 2, while in both scenarios with pre-sorted data, variant 2 is significantly more energy-efficient. As Table 7 shows, the energy efficiency of Quick Sort variant 1 significantly decreases when the data is 50% pre-sorted. For the *large* problem size on server A, the performance drops drastically, to only 16.95 Kb sorted per second. This strong decrease in performance might be due to the Quick Sort implementation with memory-based swapping getting close to its worst-case execution time of n^2 in this configuration. When the data is pre-sorted for 50%, the energy efficiency of Merge Sort variant 1 decreases by a large margin, compared to the case of 20% pre-sorted and non-pre-sorted data.

In our experiments, we have seen a significant difference between the implementation variants' mean energy efficiency in most circumstances, as indicated by the fact that the respective confidence intervals of the energy efficiencies of the variants do not overlap. A notable exception is Heap Sort on server B, where in both scenarios with pre-sorted data and across all problem sizes, no significant differences between the implementation variants could be observed. On server A, the energy efficiency of the Heap Sort implementation variants does not seem to be significantly different on problem size *small* and *large*. Overall, the number of overlaps is largest in our experiments when the data is 50% pre-sorted.

The results further demonstrate that the relationship between performance and energy efficiency is non-trivial. For instance, in both scenarios with pre-sorted data, the performance of Heap Sort on server B does not vary much (approximately 50000 kb per second

Table 5: Mean energy efficiency for both implementation variants in sorted kB per Joule. [17]

	Server	Problem Size	Variant 1 $Efficiency_E$	Variant 1 95% CI	Variant 2 $Efficiency_E$	Variant 2 95% CI	V1 and V2 Overlap
Merge Sort	A	Small	3 057 140.43	68 234.82	401 219.46	1867.08	✗
		Medium	2 874 856.35	607 613.69	393 527.45	20 993.10	✗
		Large	3 406 575.21	109 261.19	376 680.40	41 104.92	✗
	B	Small	3 554 848.68	702 507.71	322 805.22	1190.23	✗
		Medium	3 946 797.65	572 690.29	318 950.61	3342.17	✗
		Large	4 263 224.30	261 687.77	317 969.18	3109.83	✗
Heap Sort	A	Small	156 507.34	9793.82	156 538.77	2647.30	✓
		Medium	159 005.12	491.70	156 300.54	1164.83	✗
		Large	156 701.09	2314.56	153 358.12	4113.29	✓
	B	Small	112 808.83	165.06	107 845.36	110.66	✗
		Medium	112 207.69	350.81	107 072.05	135.36	✗
		Large	111 542.68	403.48	106 421.56	474.03	✗
Quick Sort	A	Small	712 085.62	2818.19	581 176.35	49 584.84	✗
		Medium	687 025.63	80 615.43	583 890.48	44 280.08	✓
		Large	689 021.31	74 343.39	604 550.75	6370.04	✗
	B	Small	539 591.67	6980.36	424 654.50	5686.78	✗
		Medium	544 565.18	7796.15	420 779.36	3926.93	✗
		Large	545 010.71	5673.11	424 777.04	3415.20	✗
Insertion Sort	A	Small	1571.49	23.95	1299.39	13.67	✗
		Medium	1136.63	11.74	851.60	5.85	✗
		Large	819.89	5.01	600.31	2.20	✗
	B	Small	1565.20	108.45	1328.51	29.46	✗
		Medium	1017.02	24.45	829.03	11.50	✗
		Large	711.04	11.77	578.79	6.98	✗
Bubble Sort	A	Small	464.87	2.79	447.34	2.23	✗
		Medium	249.44	1.82	237.70	0.37	✗
		Large	165.24	1.01	157.21	0.72	✗
	B	Small	368.83	2.20	349.39	2.87	✗
		Medium	192.95	0.88	181.93	1.16	✗
		Large	127.86	0.52	121.80	1.31	✗
Selection Sort	A	Small	571.07	3.53	901.45	5.29	✗
		Medium	314.75	1.36	537.04	3.14	✗
		Large	211.29	1.31	369.80	0.79	✗
	B	Small	497.21	2.60	837.31	23.62	✗
		Medium	264.64	1.39	477.16	4.57	✗
		Large	176.83	1.09	320.51	1.09	✗

in both scenarios); the energy efficiency, on the other hand, drastically decreases when the data is 50% pre-sorted. In addition, for both implementations of Heap Sort we observe a higher mean performance on the large problem size in comparison to the medium problem size but a lower mean energy efficiency on server B for both the 20% pre-sorted data and the 50% pre-sorted data. For server A, the same observation can be made for both implementations of Heap Sort on the 50% pre-sorted data. Another example is Quick Sort, for which this effect can be seen for implementation variant 2 on server B in the scenario with 50% of the data being pre-sorted and on server A when the data is 20% pre-sorted. In summary, the results demonstrate that the choice of algorithm and its implementation, as well as the degree to which the data is pre-sorted, can have significant impact on the energy consumption.

5 RECOMMENDATIONS

Based on our analysis in Section 4, we compiled some basic recommendations for Merge Sort, Heap Sort, and Quick Sort. Bubble Sort, Selection Sort, and Insertion Sort are the least energy-efficient across all experiments, which can be attributed to their average

Table 6: Mean energy efficiency for both C implementation variants in sorted kB per Joule with approximately 20% of data being pre-sorted. In addition, we report the mean performance in sorted kB per second.

	Server	Problem Size	Variant 1 Performance	Variant 1 $Efficiency_E$	Variant 1 95% CI	Variant 2 Performance	Variant 2 $Efficiency_E$	Variant 2 95% CI	V1 and V2 Overlap
Merge Sort	A	Small	604 729.91	7 226 675.77	258 008.91	128 675.95	633 450.26	13 732.42	✗
		Medium	639 617.16	7 608 674.38	629 193.97	128 370.32	599 772.35	12 512.02	✗
		Large	660 717.49	7 862 375.87	355 132.14	128 639.72	564 518.74	6565.78	✗
	B	Small	438 093.03	4 645 249.78	1 542 776.39	100 503.58	1 056 276.30	483 836.50	✗
		Medium	549 780.03	5 256 787.92	702 389.74	125 422.31	1 236 522.83	19 817.66	✗
		Large	625 780.94	5 837 467.04	56 078.35	125 647.86	1 255 210.35	6064.84	✗
Heap Sort	A	Small	49 630.19	149 080.29	474.10	48 316.82	141 192.27	1111.09	✗
		Medium	48 966.30	143 856.52	332.55	47 786.90	137 083.77	71.30	✗
		Large	48 936.58	142 315.77	29.02	47 658.96	135 156.26	303.32	✗
	B	Small	48 693.01	263 214.81	2934.59	49 787.76	261 954.56	643.24	✓
		Medium	47 716.74	244 977.11	7146.92	49 140.34	246 847.89	3668.94	✓
		Large	47 957.82	239 117.31	4751.15	49 178.39	241 023.68	2512.59	✓
Quick Sort	A	Small	36 052.76	124 860.80	141.21	188 511.70	2 173 849.85	138 240.14	✗
		Medium	25 216.41	96 432.52	2897.99	169 825.00	1 601 543.06	699 558.05	✗
		Large	39 221.62	132 254.86	112.03	192 946.51	1 600 880.31	55 879.56	✗
	B	Small	30 890.77	151 402.67	1067.95	172 159.51	1 726 402.11	294 948.90	✗
		Medium	37 360.86	186 512.57	6769.22	177 049.82	1 756 853.85	243 615.47	✗
		Large	36 372.68	178 620.21	3598.98	177 167.79	1 769 536.51	240 796.61	✗
Insertion Sort	A	Small	299.32	3633.81	29.13	283.59	3468.03	69.33	✗
		Medium	301.58	3802.35	51.70	262.69	3321.75	73.73	✗
		Large	251.02	1334.67	93.70	210.64	923.29	43.88	✗
	B	Small	275.17	2588.00	17.98	232.92	2287.32	62.15	✗
		Medium	263.77	2565.22	49.07	204.55	2028.40	6.56	✗
		Large	215.96	2118.34	29.96	158.50	1423.02	29.65	✗
Bubble Sort	A	Small	121.23	1542.12	14.72	116.12	1344.53	129.24	✗
		Medium	76.35	238.75	2.07	71.82	219.83	3.86	✗
		Large	53.06	139.73	0.26	49.90	130.33	2.31	✗
	B	Small	113.31	1110.36	29.34	110.80	1066.79	41.94	✓
		Medium	70.34	407.15	3.13	68.27	379.59	3.32	✗
		Large	49.11	201.72	1.36	47.76	189.98	0.55	✗
Selection Sort	A	Small	151.44	1906.25	18.89	218.06	2695.36	12.97	✗
		Medium	101.58	360.46	7.17	169.44	914.70	13.54	✗
		Large	72.44	201.02	2.01	125.68	407.14	2.13	✗
	B	Small	131.92	1288.17	22.69	186.60	1813.79	3.43	✗
		Medium	86.93	600.13	13.11	139.15	1363.93	22.40	✗
		Large	61.31	270.88	2.48	102.36	578.08	5.79	✗

time complexity of n^2. Therefore, we do not recommend the use of these algorithms and they will not be considered in the following guidelines.

Merge Sort. Preferable over all other selected algorithms in all investigated scenarios, provided that memory is no constraint. The implementation should use pre-allocation of memory instead of dynamic allocation. It also scales well with problem size.

Quick Sort. Preferable over Heap Sort if memory is not a hard constraint. When the data is not pre-sorted, use an implementation variant with memory-based swapping. When the data is already partially sorted, use an implementation variant with pointer-based swapping instead of memory-based swapping.

Heap Sort. Preferable only over Merge Sort and Quick Sort if memory is a hard constraint.

Table 7: Mean energy efficiency for both C implementation variants in sorted kB per Joule with approximately 50% of data being pre-sorted. In addition, we report the mean performance in sorted kB per second.

Algorithm	Server	Problem Size	Variant 1 Performance	Variant 1 $Efficiency_E$	Variant 1 95% CI	Variant 2 Performance	Variant 2 $Efficiency_E$	Variant 2 95% CI	V1 and V2 Overlap
Merge Sort	A	Small	495 898.52	7 481 817.33	316 176.56	137 745.20	615 232.08	31 679.89	✗
		Medium	554 983.03	8 103 984.30	248 482.84	138 404.66	593 207.44	9473.49	✗
		Large	580 883.41	8 655 502.10	292 472.06	139 417.39	569 874.89	7966.55	✗
	B	Small	441 877.63	1 345 769.63	269 197.59	115 270.90	353 258.51	62 046.30	✗
		Medium	465 916.70	1 701 845.85	431 471.65	130 627.41	477 743.71	115 252.22	✗
		Large	510 472.94	1 363 307.92	230 911.01	129 890.45	337 188.47	12 612.42	✗
Heap Sort	A	Small	55 955.77	163 561.83	10 384.19	55 470.05	158 657.44	709.38	✓
		Medium	57 356.30	163 088.19	312.09	55 050.51	154 190.05	463.42	✗
		Large	57 387.12	161 413.85	473.14	55 183.76	153 170.74	460.74	✗
	B	Small	49 044.07	137 836.83	14 544.12	53 995.83	155 754.17	19 375.27	✓
		Medium	51 863.27	187 087.99	46 987.28	53 147.49	195 932.26	50 964.51	✓
		Large	52 111.77	137 480.05	9086.18	53 501.63	141 023.65	9401.27	✓
Quick Sort	A	Small	325.42	1241.72	224.76	198 017.67	1 551 259.82	21 823.89	✗
		Medium	4081.32	14 560.23	8.23	200 020.05	1 426 626.22	47 272.31	✗
		Large	16.95	41.09	2.17	205 764.92	451 937.76	5469.30	✗
	B	Small	8834.01	49 010.39	24 110.42	160 797.63	500 496.84	49 394.98	✗
		Medium	1406.19	50 197.02	36 052.31	155 899.85	444 691.99	75 834.45	✗
		Large	5364.61	51 511.26	33 082.98	184 972.21	485 434.48	38 230.55	✗
Insertion Sort	A	Small	298.78	4224.64	18.11	284.32	4017.79	605.31	✓
		Medium	297.57	4375.40	272.73	261.32	2344.11	227.21	✗
		Large	248.96	1107.18	22.35	206.27	800.51	26.40	✗
	B	Small	273.69	738.98	49.17	232.12	643.33	41.24	✗
		Medium	263.78	937.34	198.98	204.93	734.01	156.62	✓
		Large	215.05	556.70	1.64	159.09	413.55	1.46	✗
Bubble Sort	A	Small	118.93	847.11	31.72	114.07	738.51	34.12	✗
		Medium	74.41	221.37	1.09	70.76	208.82	1.28	✗
		Large	51.63	134.41	0.08	49.09	127.43	0.79	✗
	B	Small	112.70	304.15	10.08	109.75	296.47	9.65	✓
		Medium	70.27	231.02	35.87	68.27	228.03	37.44	✓
		Large	49.10	128.79	0.15	47.66	125.02	0.23	✗
Selection Sort	A	Small	150.13	2156.10	50.91	215.70	3664.01	185.54	✗
		Medium	99.18	325.75	2.75	167.89	740.03	11.03	✗
		Large	69.84	156.39	0.99	122.73	273.38	0.14	✗
	B	Small	131.93	415.94	59.48	185.82	584.52	87.88	✗
		Medium	86.92	226.28	0.10	139.08	360.12	1.47	✗
		Large	61.34	162.99	1.77	102.42	271.75	3.33	✗

6 THREATS TO VALIDITY

In this section, we will briefly discuss the threats to validity. They are addressed in the order of their severity as rated by the authors.

Number of repetitions. Every combination of server, algorithm, problem size, and degree to which the data is pre-sorted, was executed and measured five times. Using a larger number of repetitions per configuration could reduce the variance and provide more expressive statistics. For instance, the confidence intervals might be tightened by a larger number of repetitions that could lead to some of the detected overlaps of the confidence intervals of the different implementation variants being resolved. Even though this could affect some of the conclusions of our analysis, our presented guidelines still would be useful for selecting a good algorithm, even if not the optimal one in every situation.

Programming language selection. For the implementation of the selected algorithms, we selected the C programming language, given its wide-spread use and supposed prevalence in the future. We are aware that interpreted languages might behave differently, and that the compilation process introduces optimizations. However, we argue that compiler optimizations are part of a realistic development scenario and are widespread in use by C developers.

Degree to which the data is pre-sorted. In our study, we analyzed the energy efficiency of the selected algorithms under three different conditions with respect to the degree to which the data is pre-sorted. In the context of this work, we did not choose a larger number of scenarios due to the already large number of configurations to be tested and the resulting long duration of the execution of our experiments. Nevertheless, we consider the selected pre-conditions realistic reference values and have shown that this property of the data not only affects the energy efficiency of the sorting algorithms, but also that different algorithms can be better suited for different scenarios.

Problem size selection. We selected three different problem sizes for the set of integers to be sorted by the algorithms. The sizes were selected in accordance with [17] so that the individual sets fit into the memory of the selected servers to ensure stable measurements and to ensure the feasibility of the experiments with respect to runtime. Given the quadratic time complexities of some of the selected algorithms, different problem sizes were chosen for these algorithms. Given that we compare the normalized and not the absolute energy efficiencies of the algorithms, we do not consider the selection of different problem sizes a threat to the validity of our conclusions.

Limited number of server configurations. We conducted our experiments on two state-of-the-art servers, with CPUs from two different major manufacturers. Even though we consider these representative for x86 systems used in today's cloud data centers, using different hardware could yield results that differ from ours.

7 CONCLUSION

Energy efficiency is an increasing concern in the IT sector. Taking energy efficiency into account when implementing tasks that are executed very frequently can make a significant difference in the overall energy consumption. In this work, we analyzed the energy efficiency of six well-known sorting algorithms in two implementation variants and with partially sorted input data. While time complexity can be used as an important first pointer of which algorithm to choose, energy efficiency can vary significantly among algorithms with similar time complexity. We observed that the degree to which the data is pre-sorted, as well as small changes in implementation, can significantly impact energy consumption. Future work will focus on further characterizing such relationships as well as the influence the available hardware resources and their usage by the individual algorithms have on energy efficiency. We hope this work will help with the selection of an energy-efficient sorting algorithm and will inspire researchers and practitioners to investigate and to consider the energy efficiency of often executed tasks.

REFERENCES

[1] Karan Aggarwal, Abram Hindle, and Eleni Stroulia. 2015. GreenAdvisor: A tool for analyzing the impact of software evolution on energy consumption. In *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 311–320. https://doi.org/10.1109/ICSM.2015.7332477

[2] Karan Aggarwal, Chenlei Zhang, Joshua Charles Campbell, Abram Hindle, and Eleni Stroulia. 2014. The Power of System Call Traces: Predicting the Software Energy Consumption Impact of Changes. In *Proceedings of 24th Annual International Conference on Computer Science and Software Engineering* (Markham, Ontario, Canada) *(CASCON '14)*. IBM Corp., USA, 219–233.

[3] Anders Andrae and Tomas Edler. 2015. On Global Electricity Usage of Communication Technology: Trends to 2030. *Challenges* 6 (04 2015), 117–157. https://doi.org/10.3390/challe6010117

[4] Christian Bunse, Hagen Höpfner, Suman Roychoudhury, and Essam Mansour. 2009. Choosing the "Best" Sorting Algorithm for Optimal Energy Consumption. *ICSOFT 2009 - 4th International Conference on Software and Data Technologies, Proceedings* 2, 199–206.

[5] Eugenio Capra, Chiara Francalanci, and Sandra A. Slaughter. 2012. Is software "green"? Application development environments and energy efficiency in open source applications. *Information and Software Technology* 54, 1 (2012), 60 – 71. https://doi.org/10.1016/j.infsof.2011.07.005

[6] Tej Bahadur Chandra, VK Patle, and Sanjay Kumar. 2013. New horizon of energy efficiency in sorting algorithms: green computing. In *Proceedings of National Conference on Recent Trends in Green Computing. School of Studies in Computer in Computer Science & IT, Pt. Ravishankar Shukla University, Raipur, India.* 24–26.

[7] Nikolas Herbst. 2018. *Methods and Benchmarks for Auto-Scaling Mechanisms in Elastic Cloud Environments.* Ph.D. Dissertation. University of Würzburg, Germany.

[8] Yichao Jin, Yonggang Wen, and Qinghua Chen. 2012. Energy Efficiency and Server Virtualization in Data Centers: An Empirical Investigation. In *2012 IEEE Conference on Computer Communications Workshops*. 133–138.

[9] Samuel Kounev, Klaus-Dieter Lange, and Jóakim von Kistowski. 2020. *Systems Benchmarking* (1 ed.). Springer International Publishing. https://doi.org/10.1007/978-3-030-41705-5

[10] Klaus-Dieter Lange. 2009. The Next Frontier for Power/Performance Benchmarking: Energy Efficiency of Storage Subsystems. In *Proceedings of the 2009 SPEC Benchmark Workshop on Computer Performance Evaluation and Benchmarking* (Austin, TX). Springer-Verlag, Berlin, Heidelberg, 97–101. https://doi.org/10.1007/978-3-540-93799-9_6

[11] Ding Li, Shuai Hao, William G. J. Halfond, and Ramesh Govindan. 2013. Calculating Source Line Level Energy Information for Android Applications. In *Proceedings of the 2013 International Symposium on Software Testing and Analysis* (Lugano, Switzerland) *(ISSTA 2013)*. Association for Computing Machinery, New York, NY, USA, 78–89. https://doi.org/10.1145/2483760.2483780

[12] Candy Pang, Abram Hindle, Bram Adams, and Ahmed E. Hassan. 2016. What Do Programmers Know about Software Energy Consumption? *IEEE Software* 33, 3 (May 2016), 83–89. https://doi.org/10.1109/MS.2015.83

[13] Abhinav Pathak, Y. Charlie Hu, and Ming Zhang. 2012. Where is the Energy Spent inside My App? Fine Grained Energy Accounting on Smartphones with Eprof. In *Proceedings of the 7th ACM European Conference on Computer Systems* (Bern, Switzerland) *(EuroSys '12)*. Association for Computing Machinery, New York, NY, USA, 29–42. https://doi.org/10.1145/2168836.2168841

[14] Gustavo Pinto and Fernando Castor. 2017. Energy efficiency: A new concern for application software developers. *Commun. ACM* 60 (11 2017), 68–75. https://doi.org/10.1145/3154384

[15] M. Rashid, L. Ardito, and M. Torchiano. 2015. Energy Consumption Analysis of Algorithms Implementations. In *2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 1–4. https://doi.org/10.1109/ESEM.2015.7321198

[16] Huigui Rong, Haomin Zhang, Sheng Xiao, Canbing Li, and Chunhua Hu. 2016. Optimizing energy consumption for data centers. *Renewable and Sustainable Energy Reviews* 58 (2016), 674–691. https://doi.org/10.1016/j.rser.2015.12.283

[17] Norbert Schmitt, Supriya Kamthania, Nishant Rawtani, Luis Mendoza, Klaus-Dieter Lange, and Samuel Kounev. 2021. Energy-Efficiency Comparison of Common Sorting Algorithms. In *2021 29th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. 1–8. https://doi.org/10.1109/MASCOTS53633.2021.9614299

[18] Lide Zhang, Birjodh Tiwana, Robert P. Dick, Zhiyun Qian, Z. Morley Mao, Zhaoguang Wang, and Lei Yang. 2010. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *2010 IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. 105–114.