АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ

Тема 6 Слияние-распределение

Тема 6 Слияние-распределение

Outline

1. Сортировка слиянием
2. Внешняя сортировка
3. Сортировка подсчетом
4. Инженерный подход
5. Поразрядная сортировка

Lecturer ScD, Professor K. Smelyakov

Тема 6 Слияние-распределение

1 Сортировка слиянием

Pump up U·MIND·

Идея алгоритма

Цель лекции – рассмотрение эффективных алгоритмов сортировки слиянием и распределением. Начнем с алгоритма сортировки слиянием.

Массив $\boldsymbol{a}_{1\times m}$ чисел итерационно, $i = 0,1,2, \ldots$, разбивается на $n = \left[m/2^i\right] + \left[1\ if\ (m\ mod\ 2^i) \neq 0\right]$ массивов (на нулевой итерации – это просто числа, на первой – массивы не более чем по 2 элемента, на второй – не более чем по 4 и т.д.), где «/» – целочисленное деление. На каждой итерации (для фиксированного n) производится слияние смежных массивов так. Последовательно сравниваются элементы $a_i \leq a_j$ смежных массивов в крайней левой позиции, меньший из них выписывается; когда в одном массиве элементы исчерпаются, элементы из другого дописывают в конец. Пример.

Иллюстрация работы алгоритма

Массив с числом элементов кратным степени числа 2

1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
503	087	512	061	908	170	897	275	653	426	154	509	612	677	765	703

2

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
087	503	061	512	170	908	275	897	426	653	154	509	612	677	703	765

3

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
061	087	503	512	170	275	897	908	154	426	509	653	612	677	703	765

4

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
061	087	170	275	502	512	897	908	154	426	509	612	653	277	703	765

5

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
061	087	154	170	275	426	503	509	512	612	653	677	703	765	897	908

Иллюстрация работы алгоритма

Массив с числом элементов не кратным степени числа

2

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
503	087	512	061	908	170	897	275	653	426	154	509	612	677	765	

1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
087	503	061	512	170	908	275	897	426	653	154	509	612	677	765	

2

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
061	087	503	512	170	275	897	908	154	426	509	653	612	677	765	

3

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
061	087	170	275	503	512	897	908	154	426	509	612	653	677	765	

4

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
061	087	154	170	275	426	503	509	512	612	653	677	765	897	908	

5

Иллюстрация работы алгоритма

Упорядочить массив по возрастанию

1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
12	34	51	05	78	64	33	91	13	44	01	50	23	12	25	81

2

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

3

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

4

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

5

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Модификации

1) Сортировка по убыванию (если использовать отношение порядка $a_i \geq a_{i+1}$).

2) Вставка одной упорядоченной последовательности в другую за 1-ин проход. Двухпутевое слияние – базовый алгоритм для **внешней сортировки** уже упорядоченных массивов. Но обо всем по порядку.

Вывод оценок трудоемкости

Давайте получим оценки трудоемкости сортировки слиянием. **Кто желает помочь? Есть ли зависимость по данным, есть ли лучший и худший случаи?**

Оценки трудоемкости

Сортировка двухпутевым слиянием имеет наилучший (массив уже упорядочен) и наихудший (производится максимальное количество сравнений элементов) случаи. Однако же оценки трудоемкости обработки таких ситуаций очень близки.

Поэтому используя стандартный подход трудоемкость двухпутевого слияния оценивают по порядку величины так

$$T(n) = n \cdot \ln(n).$$

Важный частный случай:
Как изменится эта оценка для слияния двух отсортированных массивов?

Достоинства и недостатки

Достоинства:

- применим для любых чисел;
- практически нет разницы между лучшим и худшим случаями → стабильность оценки, отсутствие неоднозначности сравнительного анализа;
- достаточно высокая эффективность реализации;
- вставка упорядоченной последовательности в другую за один проход;
- можно применять как для внутренней, так и для внешней сортировки упорядоченных массивов; в этом смысле
- хорошо сочетается с подкачкой и кешированием памяти.

Достоинства и недостатки

Недостатки:

- необходимость дополнительной оперативной памяти в связи с использованием дополнительного массива приемника, что в наше время недостаток небольшой.

Тема 6 Слияние-распределение

2 Внешняя сортировка

Pump up U·MIND·

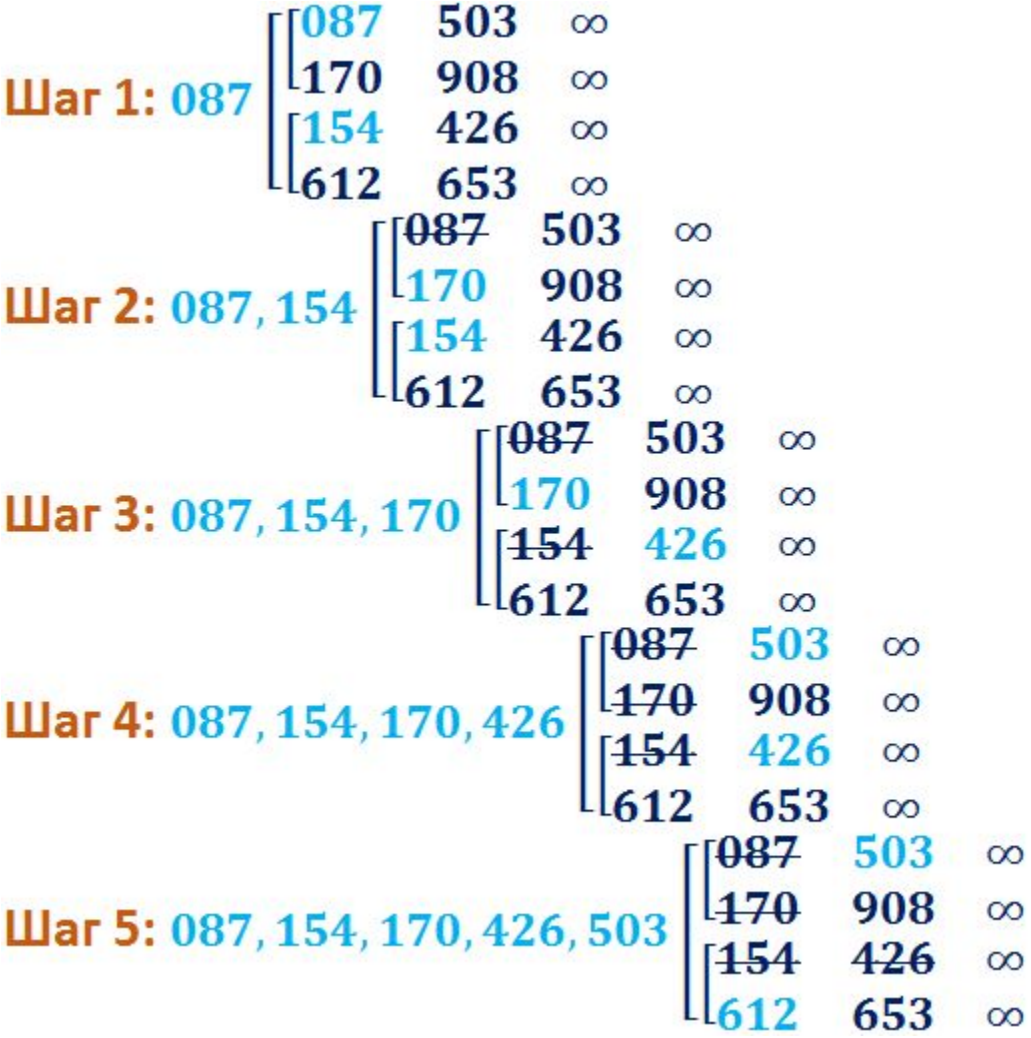
Вариации на тему слияния

Итак, допустим, что сортируемые данные (и программный код, используемый для их обработки) не помещаются в оперативной памяти. Что же делать в такой ситуации? В такой ситуации применяют такую трехэтапную схему сортировки:

- разбивают массив на части, которые помещаются в оперативной памяти;
- сортируют эти части массива в оперативной памяти (применяя определенную подкачки и кеширования памяти);
- производят слияние отсортированных массивов.

Два основных метода слияния отсортированных массивов – это двух- и многопутевое слияние (Кнут Д., 295 с.).

Четырехпутевое слияние – пример



Четырехпутевое слияние – задание

Дано: 4 упорядоченных массива на диске. Объем ОЗУ – 20 элементов. **Отсортировать** массивы 4-путевым слиянием.

12	17	33	45	51	60	66	90
1	5	7	44	56	78	87	89
20	25	34	77	79	88	98	99
3	12	19	23	37	65	79	85

– Исходные данные на диске

Сортировка в ОЗУ:

				12	17	33	45
				1	5	7	44
				20	25	34	77
				3	12	19	23

Отсортированные данные на диске

Как часто будем гонять данные: диск ↔ ОЗУ?

Тема 6 Слияние-распределение

3 Сортировка подсчетом

Pump up U·MIND·

Идея алгоритма

На входе задан массив $\boldsymbol{a}_{1\times m}$ целых неотрицательных чисел в диапазоне $[0,\dots,n-1]$: $a_i \in [0,\dots,n-1]$.

Шаг 1 – посчитать сколько раз в массиве $\boldsymbol{a}$ встречаются числа из диапазона $[0,\dots,n-1]$; результаты подсчета внести в ячейки массива счетчика $\boldsymbol{c}_{1\times n}$ (индексы ячеек отвечают самим числам диапазона $[0,\dots,n-1]$). Далее рассматривается массив счетчик $\boldsymbol{c}_{1\times n}$.

Шаг 2 – соответствующее количество раз c_i выводим числа i, $i = 0,\dots,n-1$, в порядке возрастания. Рассмотрим пример.

Иллюстрация работы алгоритма

Исходный массив: $\boldsymbol{a} = \{0,4,1,2,1,5,5,5,6,7,2,8,9,0,5,2,6,8,9,0\}$, $a_i \in [0, \dots, 9]$.

Счетчик $\boldsymbol{c}$ после подсчета:

Индексы	0	1	2	3	4	5	6	7	8	9
Значения	3	2	3	0	1	4	2	1	2	2

Результат: $\boldsymbol{r} = \{0,0,0,1,1,2,2,2,4,5,5,5,5,6,6,7,8,8,9,9\}$.

Модификации

1) Сортировка по убыванию (на втором шаге работы алгоритма – после подсчета – вывод чисел производится начиная с конца массива, то есть от наибольших к наименьшим).

Задание

Упорядочить массив по возрастанию.

Исходный массив $a_{1\times18}$

7	4	6	2	1	0	3	4	9	8	8	8	4	1	7	0	5	2

Счетчик $c_{1\times10}$

Индексы	0	1	2	3	4	5	6	7	8	9
Значения										

Результат сортировки $r_{1\times18}$

Как проверяем корректность сортировки?

Вывод оценок трудоемкости

Давайте оценим трудоемкость сортировки применением сортировки подсчетом. **Кто желает помочь? Есть ли зависимость по данным, есть ли лучший и худший случаи?**

Оценки трудоемкости

- Трудоемкость сортировки подсчетом

$$T(m,n) = 2 \cdot (m+n),$$

где m – размер исходного массива $\boldsymbol{a}_{1\times m}$, а n – размер массива счетчика $\boldsymbol{c}_{1\times n}$: $a_i \in [0,\dots,n-1]$.

Достоинства и недостатки

Достоинства:

- нет лучшего и худшего случая → стабильность оценки, отсутствие неоднозначности сравнительного анализа;
- при небольших значениях $n, n \ll m$, трудоемкость алгоритма по порядку величины оценивается так $T(m,n) = 2 \cdot m$ (пожалуй, одна из лучших оценок!); при значениях n сравнимых с m трудоемкость сортировки подсчетом оценивается величиной $T(m,n) = 4 \cdot m$ (тоже очень хорошо!). Если условие $n \leq m$ не выполняется, с ростом n эффективность начинает снижаться; реально уровень эффективности сортировки устанавливается сравнительным анализом с аналогами.

Достоинства и недостатки

Недостатки:

- требуется использование одного вспомогательного массива счетчика $c_{1\times n}$ (отсортированный массив после подсчета можно записать обратно в исходный массив), что в большинстве случаев не актуально при: (а) – современных объемах памяти и (б) – небольших значениях n;
- алгоритм применим только для целых положительных чисел. Но это в ряде случаев к это можно обойти. Как? За счет применения **приемов инженерного искусства**.

Тема 6 Слияние-распределение

4 Инженерный подход

Pump up U·MIND·

Отрицательные числа

Чтобы упорядочивать отрицательные целые числа нужно:

- перед сортировкой найти минимальное число массива min и, если оно отрицательное, прибавить к каждому числу a_i массива $\boldsymbol{a}$ число $|min|$: $a_i = a_i + |min|$. Все числа массива теперь будут не отрицательными
- провести сортировку
- к каждому числу a_i упорядоченного массива $\boldsymbol{a}$ прибавляем число min: $a_i = a_i + min$.

Floating Point Numbers

Если числа исходного массива представлены в формате с плавающей точкой может быть известна **абсолютная погрешность** представления чисел (задается при постановке задачи). Допустим, это погрешность $\Delta = 0.001$.

В такой ситуации умножаем числа a_i массива $\boldsymbol{a}$ на число Δ^{-1}: $a_i = a_i \cdot \Delta^{-1}$, округляем их до целого и представляем целым типом данных, сортируем, умножаем числа на Δ: $a_i = a_i \cdot \Delta$.

Зачем нужны все эти ухищрения?

Затем, что трудоемкость сортировки подсчетом удивительно низкая!

Приближенные числа

При решении инженерных задач практически всегда мы имеем дело с величинами, истинные значения которых неизвестны, а известны лишь их **приближенные значения**, то есть числа, которые выражают истинные значения некоторой величины с **погрешностью**. Собственно именно они чаще всего и записываются в формате с плавающей точкой.

И для нас важно понимать: **как работать с такими числами**?

Например, как меняется погрешность суммы и разности приближенных чисел, или как оценивать погрешность промежуточных вычислений?

Основы вычислительной математики

При работе с приближенными числами не нужно бездумно выбирать встроенный тип данных наибольшей размерности и хранить там бесполезные разряды. Напротив, нужно выяснить погрешность, выбрать адекватный тип данных и алгоритм работы с ними. А для этого, прежде всего, требуется иметь представления об:

- источниках возникновения погрешности
- видах погрешности (абсолютная и относительная) и методиках их оценивания
- значащей и верной цифре
- правилах проведения операций с приближенными числами, включая округление и отсечение.

Benefits

И все это не рутина. А необходимые нам знания.

Освоив работу с приближенными числами мы сможем:

- эффективно использовать готовые алгоритмы, оперирующие приближенными числами: экономить память и время обработки данных за счет своевременного адекватного преобразования чисел к целому типу данных и обратно
- составлять свои эффективные алгоритмы, оперирующие приближенными числами
- адекватно оценивать результаты применения алгоритмов, оперирующие приближенными числами

Пример – сортировка чисел в формате с плавающей точкой.

Литература

Работа с приближенными числами хорошо описана в книге: Демидович Б.П., Марон И.А. Основы вычислительной математики. – М.: Наука, 1966. – 664 с.

Д/З. Проработать материал:

Глава I. Приближенные числа.

Задание на лабораторную работу

Тема лабораторной работы № 5: Программная реализация алгоритмов сортировки слиянием-распределением

Задание: программная реализация алгоритмов слияния и поразрядной сортировки.

Тема 6 Слияние-распределение

5 Поразрядная сортировка

Pump up U·MIND·

Идея алгоритма

Идея алгоритма поразрядной сортировки настолько проста, что она уже на кончике языка.

Кто сформулирует идею алгоритма исходя из его названия?

С какого разряда нужно начинать сортировку?

Иллюстрация работы алгоритма

Источник / приемник

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
503	**087**	**512**	**061**	**908**	**170**	**897**	**275**	**653**	**426**	**154**	**509**	**612**	**677**	**765**	**703**

Счетчик (**итерация 1** – упорядочение по младшему разряду)

0	1	2	3	4	5	6	7	8	9	Индексы (отвечают значениям разряда числа)
1	1	2	3	1	2	1	3	1	1	Число элементов группы
0	0+1=**1**	1+1=**2**	2+2=**4**	4+3=**7**	7+1=**8**	**10**	**11**	**14**	**15**	Старт. позиция элемента группы

Приемник / источник (результат **итерации 1**)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
170	**061**	**512**	**612**	**503**	**653**	**703**	**154**	**275**	**765**	**426**	**087**	**897**	**677**	**908**	**509**

Иллюстрация работы алгоритма

Источник / приемник

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
170	**061**	**512**	**612**	**503**	**653**	**703**	**154**	**275**	**765**	**426**	**087**	**897**	**677**	**908**	**509**

Счетчик (**итерация 2** – упорядочение по младшему разряду)

0	1	2	3	4	5	6	7	8	9	Индексы (отвечают значениям разряда числа)
4	2	1	0	0	2	2	3	1	1	Число элементов группы
0	**0+4=4**	**4+2=6**	**6+1=7**	**7+0=7**	**7+0=7**	**9**	**11**	**14**	**15**	Старт. позиция элемента группы

Приемник / источник (результат **итерации 2**)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
503	**703**	**908**	**509**	**512**	**612**	**426**	**653**	**154**	**061**	**765**	**170**	**275**	**677**	**087**	**897**

Иллюстрация работы алгоритма

Источник / приемник

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
503	**703**	**908**	**509**	**512**	**612**	**426**	**653**	**154**	**061**	**765**	**170**	**275**	**677**	**087**	**897**

Счетчик (**итерация 3** – упорядочение по младшему разряду)

0	1	2	3	4	5	6	7	8	9	Индексы (отвечают значениям разряда числа)
2	2	1	0	1	3	3	2	1	1	Число элементов группы
0	0+2=**2**	2+2=**4**	4+1=**5**	5+0=**5**	5+1=**6**	**9**	**12**	**14**	**15**	Старт. позиция элемента группы

Приемник / источник (результат **итерации 3**)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
061	**087**	**154**	**170**	**275**	**426**	**503**	**509**	**512**	**612**	**653**	**677**	**703**	**765**	**897**	**908**

Сокращаем пространство имен

При начальной инициализации исходные данные записываются в массив размерностью $1 \times n$, который называют **источником**. Для записи результата сортировки требуется массив такой же размерности, который называют **приемником** (источник – откуда берут, приёмник – куда записывают).

На каждой новой итерации эти массивы меняются ролями: один источник, другой – приемник, потом наоборот.

Прием: с программной точки зрения источник и приемник удобно хранить в одном двумерном массиве размерностью $2 \times n$, чтобы использовать одно имя и указанием номера строки переключать источник и приемник между собой.

Иллюстрация работы алгоритма

Источник / приемник

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
144	**231**	**125**	**378**	**799**	**514**	**100**	**017**	**229**	**324**	**718**	**513**	**672**	**779**	**062**	**920**

Счетчик (**итерация 1** – упорядочение по младшему разряду)

0	1	2	3	4	5	6	7	8	9	
										Индексы (отвечают значениям разряда числа)
										Число элементов группы
										Старт. позиция элемента группы

Приемник / источник (результат **итерации 1**)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Иллюстрация работы алгоритма

Источник / приемник

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Счетчик (**итерация 2** – упорядочение по младшему разряду)

0	1	2	3	4	5	6	7	8	9	Индексы (отвечают значениям разряда числа)
										Число элементов группы
										Старт. позиция элемента группы

Приемник / источник (результат **итерации 2**)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Иллюстрация работы алгоритма

Источник / приемник

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Счетчик (**итерация 3** – упорядочение по младшему разряду)

0	1	2	3	4	5	6	7	8	9	Индексы (отвечают значениям разряда числа)
										Число элементов группы
										Старт. позиция элемента группы

Приемник / источник (результат **итерации 3**)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Модификации

1) Сортировка по убыванию.
2) Выбор наилучшего алгоритма выделения цифр.
3) Выбор оптимальной системы счисления.

Выделение цифры из числа

Прием 1: деление по модулю. Наиболее очевидный для нас прием выделения цифры из числа основан на использовании операции деления по модулю

$$d(j,n) = \left[n/10^{j-1}\right]\%10,$$

где n – число, d – выделяемый разряд числа, j – номер цифры числа справа, «/» – целая часть, а «%» – дробная часть от деления http://algolist.manual.ru/sort/radix_sort.php

Выделение цифры из числа

Пример: выделить цифры из числа 345:

$$d(1,345) = [345/10^{1-1}]\%10 = [345]\%10 = 5;$$
$$d(2,345) = [345/10^{2-1}]\%10 = [34]\%10 = 4;$$
$$d(3,345) = [345/10^{3-1}]\%10 = [3]\%10 = 3.$$

Задание: выделить цифры из числа 127:

$$d(1,127) =$$
$$d(2,127) =$$
$$d(3,127) =$$

Данный прием является самым трудоемким!

Выделение цифры из числа

Прием 2: Использование сдвигов и битовых масок. Если рассматривать числа, с которыми мы работаем как n-разрядные двоичные числа (что фактически отражает способ хранения чисел в памяти ЭВМ), то:

1) эти числа можно рассматривать как числа, записанные в любой системе счисления (СС), основанной на использовании степени двойки, например, в 8-ой или 16-ой СС; в настоящее время считается, что в качестве разряда лучше всего брать **один байт** (https://habrahabr.ru/post/271677/), хотя при выборе основания СС еще нужно учитывать объем выборки;

2) для выделения разрядов мы можем использовать сдвиги (как аналоги деления / умножения на 2) и наложение битовых масок для обнуления не интересующих разрядов.

Выделение цифры из числа

Примеры.

Рассматриваемое число:

01010101 10101100 (это число 21932 в десятичной СС).

1) Выделение младшего разряда в однобайтовой СС применением наложения логической маски:

01010101 10101100 AND **00000000 11111111** = **00000000 10101100**

2) Выделение старшего разряда в однобайтовой СС применением логического сдвига вправо на 8 позиций:

01010101 10101100 >> (LSR, 8) **00000000 01010101**

Базовый алгоритм выделения цифры состоит в том, чтобы в цикле: 1) наложением маски обнулить ненужные разряды, 2) применением логического сдвига вправо расположить интересующие разряды в младшем байте.

Выделение цифры из числа

Задание.

Рассматриваемое число:

11110001 10001111 (это число 6183910 в десятичной СС).

1) Выделение младшего разряда в однобайтовой СС применением наложения логической маски:

01010101 10101100 AND **00000000 11111111** =

2) Выделение старшего разряда в однобайтовой СС применением логического сдвига вправо на 8 позиций:

01010101 10101100 >> (LSR, 8) =

Выделение цифры из числа

Прием 3: Использование встроенных функций. Механизм выделения разряда: использование функции преобразования в целое число + использование параметра **radix** для определения системы счисления (https://habrahabr.ru/post/271677/).

Здесь radix хранит основание СС, например, radix = 10 даст десятичное число, radix = 16 – шестнадцатиричное и т.п. Для radix > 10 цифры после девяти представлены буквами латинского алфавита.

Например, в java script функция parseInt преобразует первый аргумент в число по указанному основанию, а если это невозможно – возвращает NaN. Особенности применения функции описаны здесь https://javascript.ru/ParseInt.

Кто хочет 50?

Нужно провести исследование вычислительной эффективности алгоритмов выделения цифры из числа и сделать доклад.
Идеи смотреть здесь: https://habrahabr.ru/post/271677/

Вывод оценок трудоемкости

Давайте оценим трудоемкость сортировки применением поразрядной сортировки. **Какие будут соображения? Есть ли зависимость по данным, есть ли лучший и худший случаи?**

Оценки трудоемкости

Трудоемкость поразрядной сортировки

$$T(m, b, k) = k \cdot (2m + b),$$

где m – число элементов сортируемого массива $\boldsymbol{a}_{1\times m}$, b – основание системы счисления (для 10-й системы счисления $b = 10$), k – число значащих разрядов сортируемых чисел.
При больших m и малых b по порядку величины трудоемкость поразрядной сортировки имеет линейный характер

$$T(m, k) = 2 \cdot k \cdot m.$$

Достоинства и недостатки

Достоинства:

- нет лучшего и худшего случая → стабильность оценки, отсутствие неоднозначности сравнительного анализа;
- линейная трудоемкость (как у сортировки подсчетом).

Недостатки:

- требует использования вспомогательных массивов, что, при современных объемах памяти редко является значимым;
- на сверхмалых выборках из-за необходимости работать со вспомогательными массивами алгоритм часто проигрывает своим аналогам по трудоемкости;
- алгоритм применим для целых положительных чисел, но, как мы уже знаем, в ряде случаев это можно обойти.

Задание на лабораторную работу

Тема лабораторной работы № 5: Программная реализация алгоритмов сортировки слиянием-распределением

Задание (опционально):

1) программная реализация поразрядной сортировки;

2) программная реализация и сравнительный анализ эффективности алгоритмов отщепления цифры (вычислять коэффициент прироста трудоемкости)

Типы	Коэффициент
Десятичный	
Двоичный	
Встроенный