Jak (prawdopodobnie) zaliczyć SW Test level Expert

Disclaimer :-)

Jak (prawdopodobnie) zaliczyć SW Test level Expert

- Jak się przygotować?
- **Krótkie** implementacje znanych algorytmów.
- Szybsze nie zawsze znaczy lepsze, koszt implementacji.
- Dynamiczna alokacja pamięci, wskaźniki – NIE!
- Cache
- Omówienie przykładowych problemów.
- Podsumowanie i gdzie znaleźć dodatkowe materiały.
- Pytanie / uwagi końcowe.
- Ankieta odnośnie szkolenia.

1. Jak się przygotować?

- Znajomość algorytmów
  - Wprowadzenie do algorytmów
    Autors: Cormen Thomas H. …
  - Niebieskie Książeczki – oi
- Potyczki Algorytmiczne: potyczki.mimuw.edu.pl
- main.edu.pl
- Ćwiczyć, ćwiczyć, ćwiczyć ….

  - zadanie zawsze powinno być wykonane

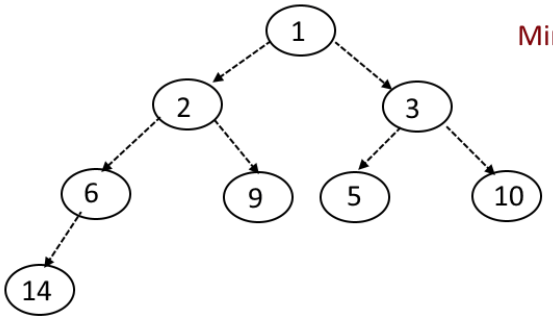
2. Krótkie implementacje znanych algorytmów

- Algorytm powinien być **krótki**
- Robi tylko to co ma robić i nic więcej
- Przykładowe algorytmy:

  - kopiec
  - sortowanie: counting sort, radix sort, insert sort, heap sort, merge sort
  - hash mapa

kopiec

```cpp
#define MAX_SIZE 100000000
int heap[MAX_SIZE];  //heap starts at heap[1]
int size = 0;

inline void swap(int i, int j) {
    int t = heap[i];
    heap[i] = heap[j];
    heap[j] = t;
}
inline bool less(int i, int j) {
    return heap[i] < heap[j];
}
void up(int start) {
    while (start > 1 && less(start, start / 2)) {
        swap(start / 2, start);
        start /= 2;
    }
}
void down(int i) {
    int right = i * 2 + 1, smaller = i * 2;
    if (right <= size && less(right, smaller))
            smaller = right;
    if (smaller <= size) {
        if (less(smaller, i)) {
            swap(i, smaller);
            down(smaller); //tail recursion! (rekurencja ogonowa)
        }
    }
}
//-----------------------
void push(int e) {
    heap[++size] = e;
    up(size);
}
int pop() {
    int r = heap[1];
    swap(1, size--);
    down(1);
    return r;
}

void make_heap() { }  //not implemented!

//----------heap sort

void hsort(int *T, int size) {
    for(int i = 0;i < size; i++) push(T[i]);
    for(int i = 0;i < size; i++) T[i] = pop();
}
```



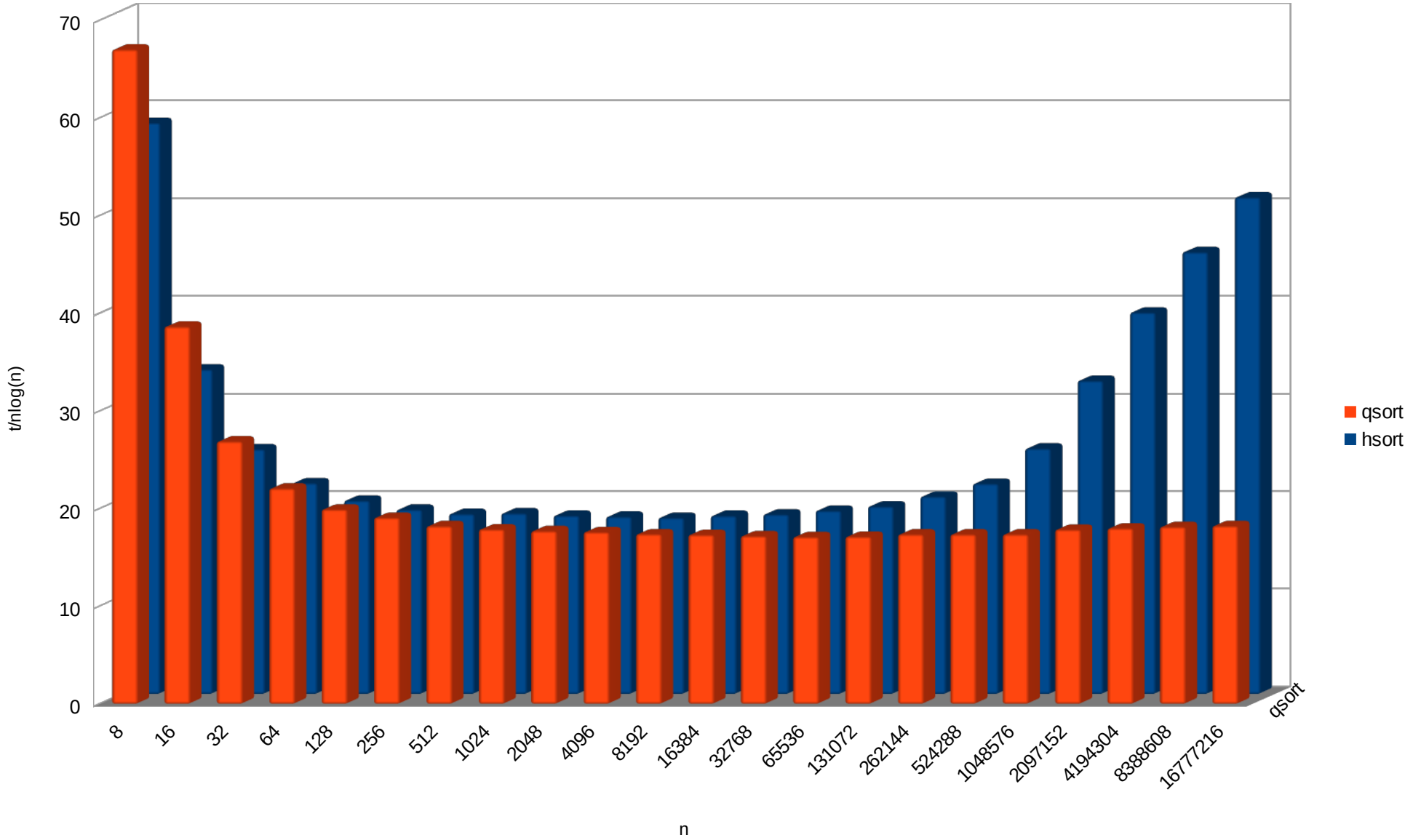
Min Heap

0	1	2	3	4	5	6	7	8
	1	2	3	6	9	5	10	14

for Node at i : Left child will be 2i and right child will be at 2i+1 and parent node will be at [i/2].

Kopiec: t/nlog(n)

vs std::qsort



Krótkie implementacje na przykładzie sortowania

- Counting sort:

```
#define MAX 20000000

static int count[MAX];

void csort(int *from, int *to, int size, int maks)
{
    for (int i = 0; i < size; i++)
        count[from[i]]++;

    for (int j = 0, i = 0; i <= maks; i++)
        while (count[i]) count[i]--, to[j++] = i;
}
```

Input Data

0	4	2	2	0	0	1	1	0	1	0	2	4	2

Count Array

0	1	2	3	4
5	3	4	0	2

Sorted Data

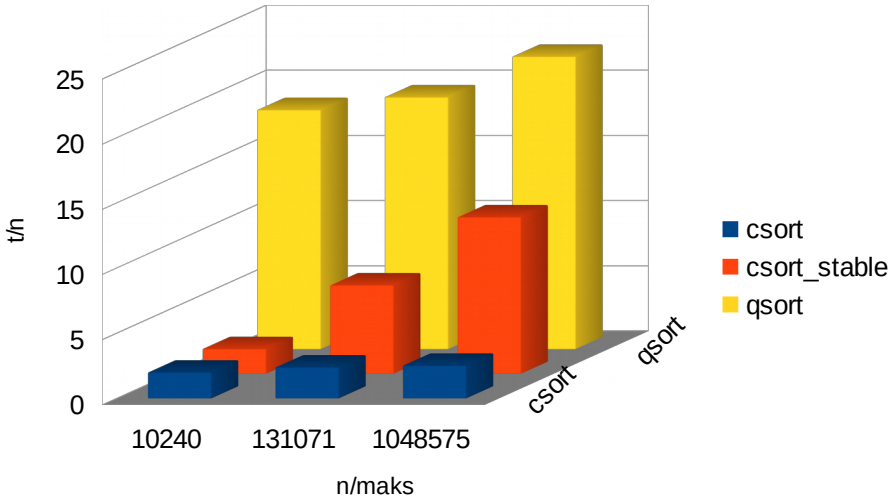
0	0	0	0	0	1	1	1	2	2	2	2	4	4

- Stable counting sort:

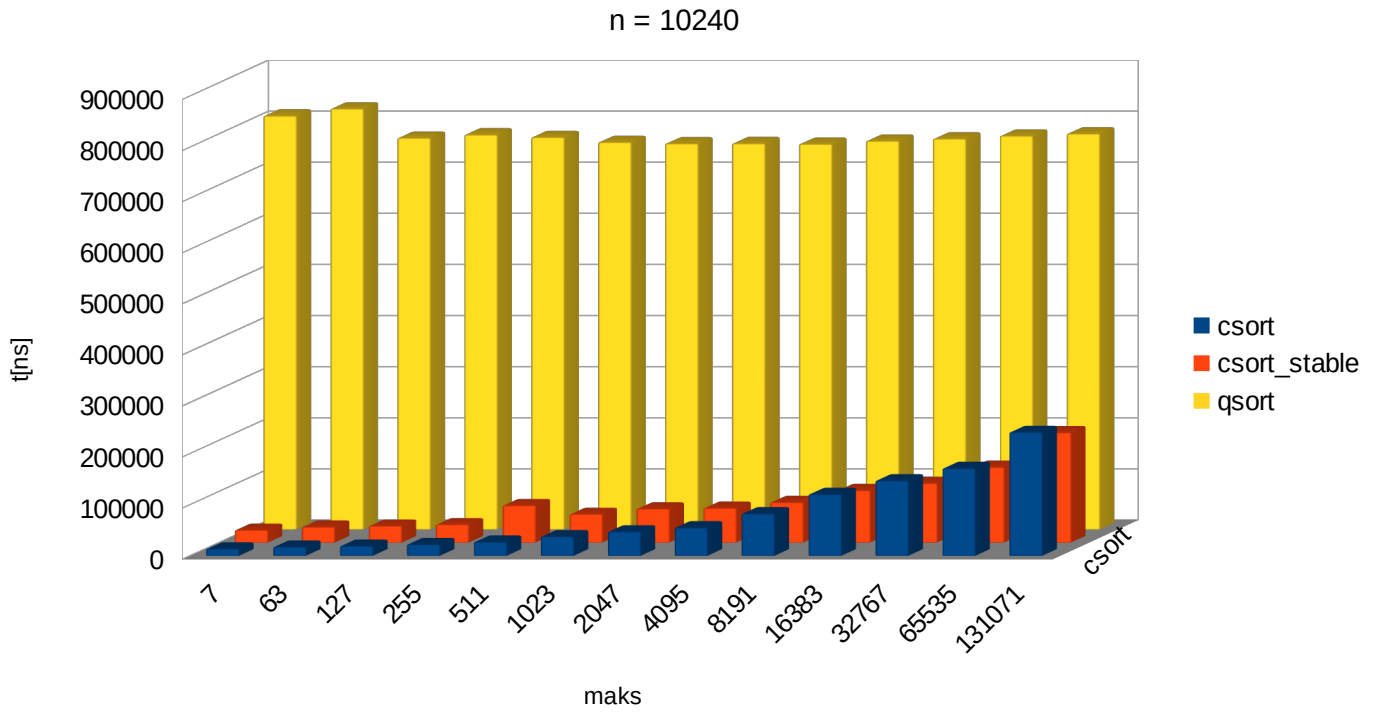
```
void csort_stable(int *from, int *to, int size, int maks)
{
    for (int i = 0; i < size; i++)
        count[from[i]]++;

    for (int i = 1; i <= maks; i++)
        count[i] += count[i - 1];
    for (int i = size - 1; i >= 0; i--)
        to[--count[from[i]]] = from[i];
    for (int i = 0; i <= maks; i++)
        count[i] = 0;
}
```

Counting sort

vs. std::qsort



n	csort	csort_ stable	qsort
10240	2,0	1,9	18,4
131071	2,4	6,8	19,3
1048575	2,5	12,0	22,5



Radix sort - sortowanie pozycyjne

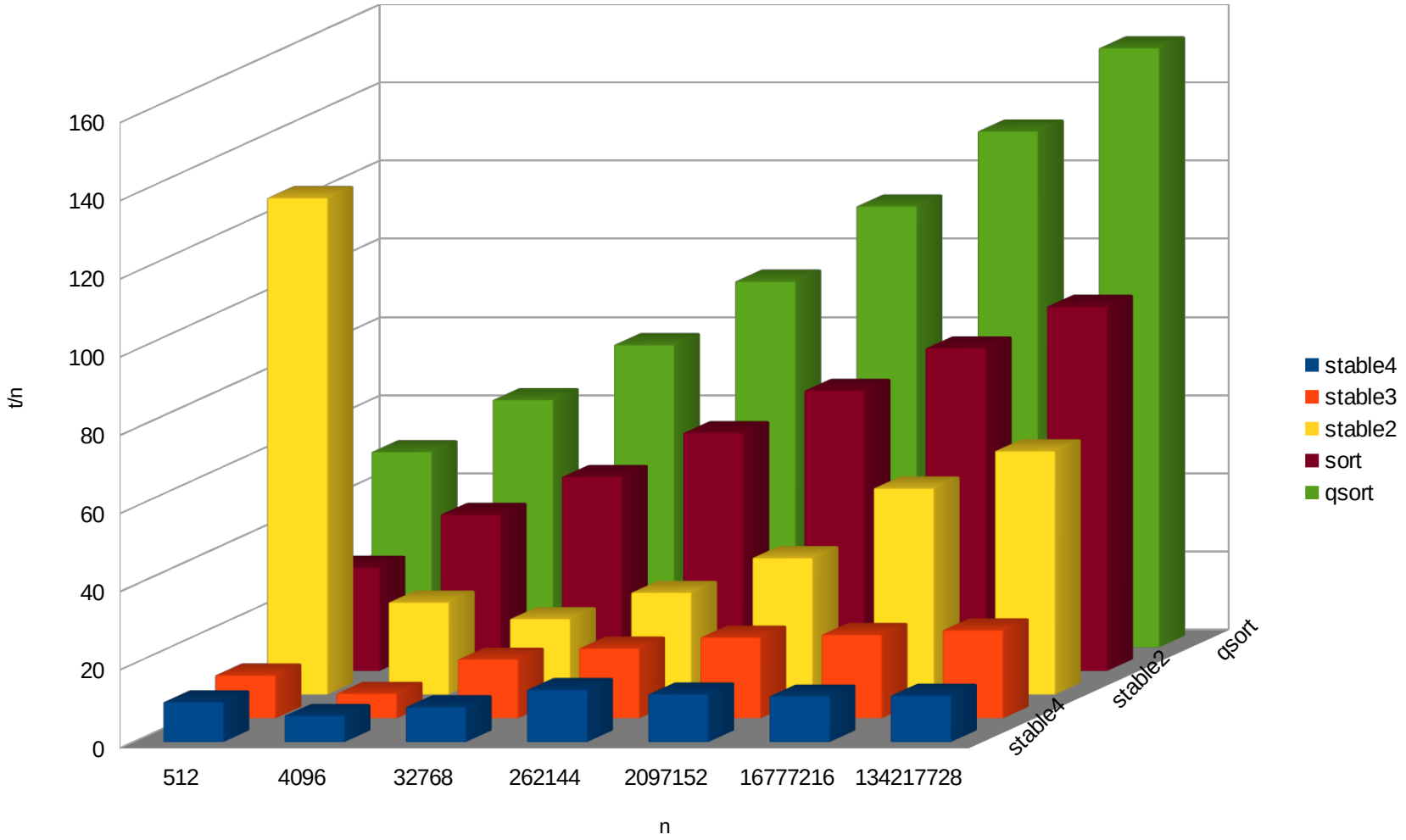
czyli „stable counting sort” po pozycjach (cyfrach)

```
#define MAX 100000

static int count[MAX];

template<int mask, int shift>
void csort_(int *from, int *to, int size)
{
    for (int i = 0; i < size; i++)
        count[(from[i] >> shift)&mask]++;

    for (int i = 1; i <= mask; i++)
        count[i] += count[i - 1];
    for (int i = size - 1; i >= 0; i--)
        to[--count[(from[i] >> shift)&mask]] = from[i];
    for (int i = 0; i <= mask; i++)
        count[i] = 0;
}

void csort_stable4(int *from, int *to, int size)
{
    csort_<255, 0>(from, to, size);
    csort_<255, 8>(to, from, size);
    csort_<255, 16>(from, to, size);
    csort_<255, 24>(to, from, size);
}

void csort_stable3(int *from, int *to, int size)
{
    csort_<2047, 0>(from, to, size);
    csort_<2047, 11>(to, from, size);
    csort_<2047, 22>(from, to, size);
}


void csort_stable2(int *from, int *to, int size)
{
    csort_<65535, 0>(from, to, size);
    csort_<65535, 16>(to, from, size);
}
```

Sortowanie dużych intów (>=0)

Radix sort int32 (>=0): t/n

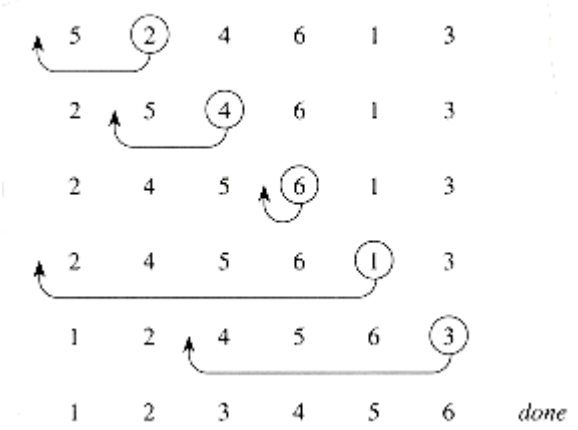
vs. std::sort() and std::qsort()



„simple” sort

- Insertion sort

```
void isort(int *T, int size)
{
    for (int j, x, i = 1; i < size; i++) {
        x = T[i];
        j = i - 1;
        while (j >= 0 && T[j] > x) {
            T[j + 1] = T[j];
            j--;
        }
        T[j + 1] = x;
    }
}
```



- Bubble sort

```
#define VAR(v, n) __typeof(n) v = (n)
#define SWAP(a, b) { VAR(t,a); a=b,b=t;}

void bsort(int *T, int size)
{
    do {
        int i, nsize = 0;
        for (i = 1; i < size; i++)
            if (T[i - 1] > T[i]) {
                SWAP(T[i - 1], T[i]);
                nsize = i;
            }
        size = nsize;
    } while (size > 0);
}
```

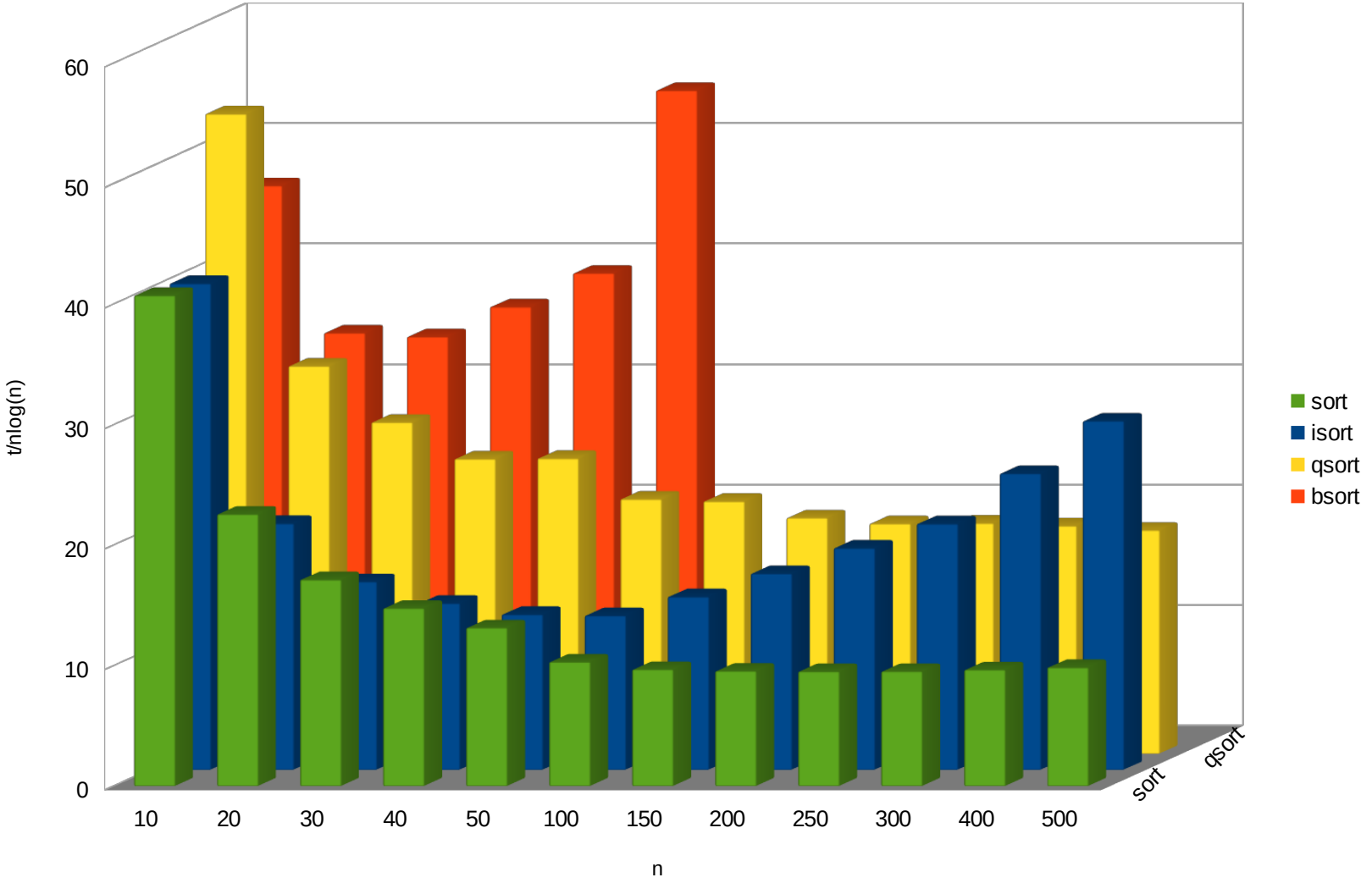
Bubble Sort Example

9, 6, 2, 12, 11, 9, 3, 7
6, 9, 2, 12, 11, 9, 3, 7
6, 2, 9, 12, 11, 9, 3, 7
6, 2, 9, 12, 11, 9, 3, 7
6, 2, 9, 11, 12, 9, 3, 7
6, 2, 9, 11, 9, 12, 3, 7
6, 2, 9, 11, 9, 3, 12, 7
6, 2, 9, 11, 9, 3, 7, 12

„simple” sort: t/nlog(n)

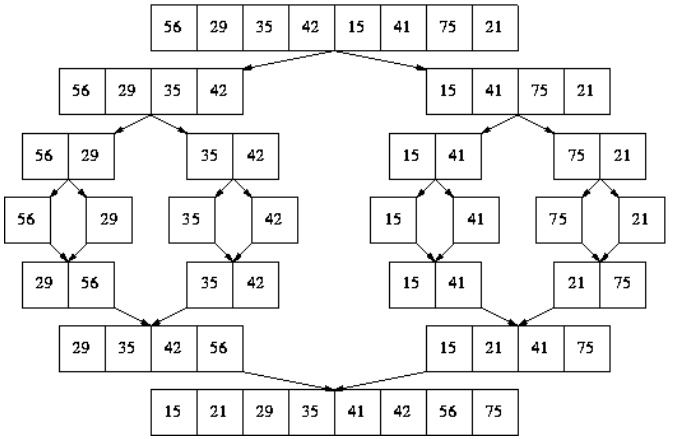
vs. std::sort() and std::qsort()

dla czytelności bsort nie jest pokazany dla n > 100



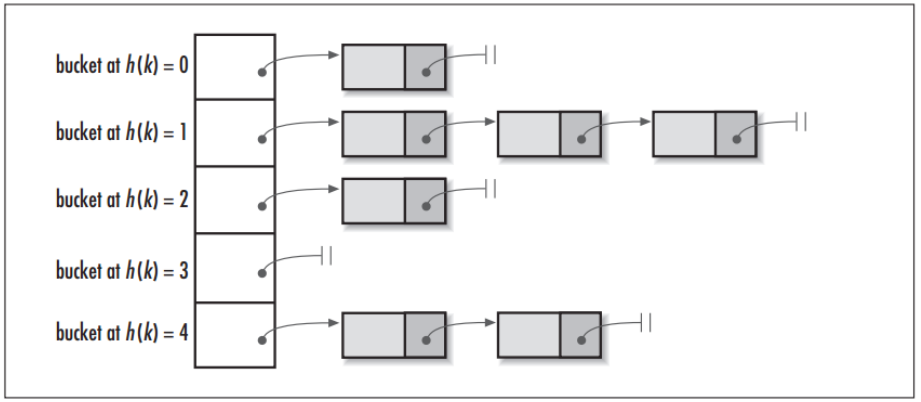
Bonus: krótka implementacja merge sort

```
3  #define MAX 100000000
4
5  int t2[MAX];
6  int *t; //should be t[MAX]
7
8  void merge_sort(int b, int e) {  // sort [b,e)
9      if (b + 1 >= e) return;
10
11     int s = (b + e) / 2;
12     merge_sort(b, s);
13     merge_sort(s, e);
14
15     int i1 = b, i2 = s;
16     for (int p = b; p < e; p++) {
17         if (i2 == e || i1 < s && t[i1] <= t[i2]) {
18             t2[p] = t[i1++];
19         } else {
20             t2[p] = t[i2++];
21         }
22     }
23     for (int p = b; p < e; p++) t[p] = t2[p];
24 }
```



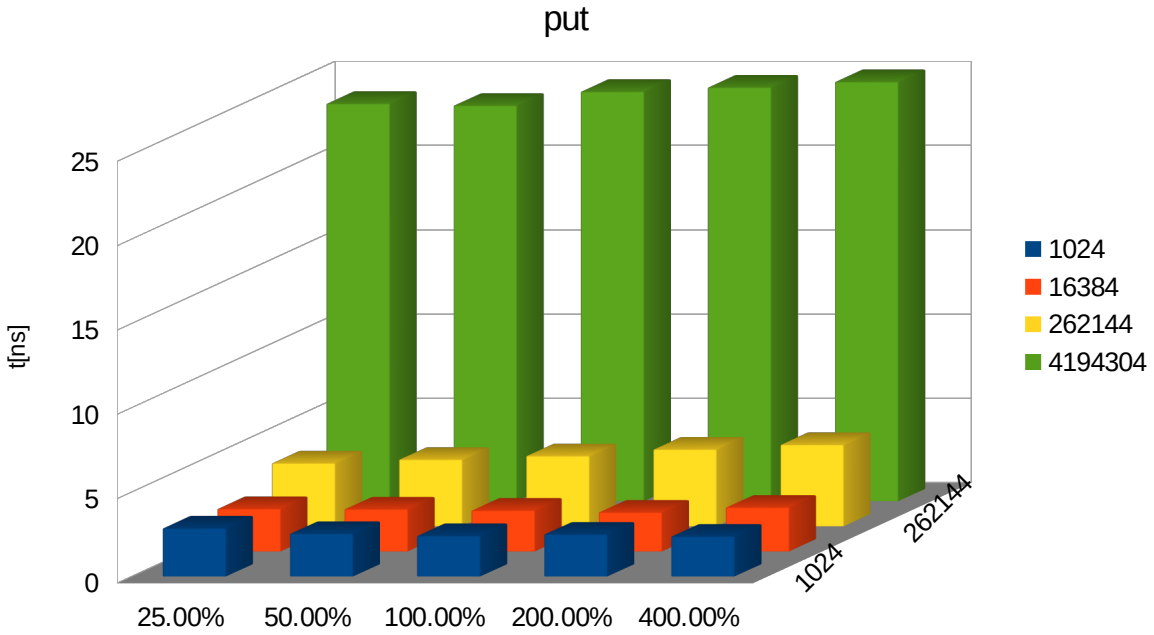
Hash map

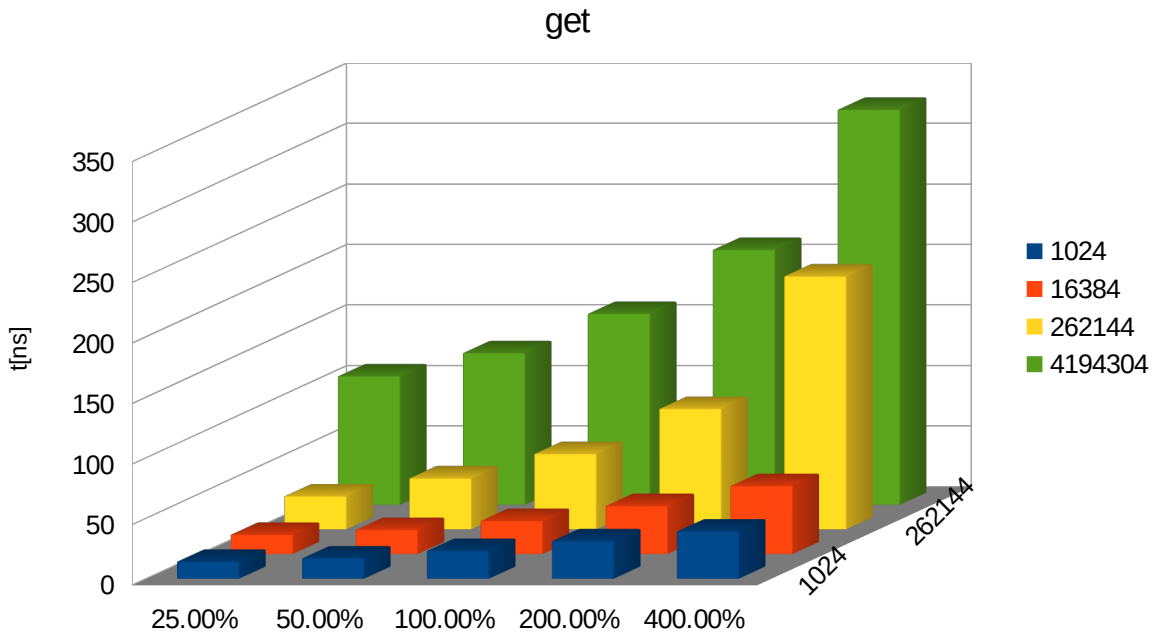
```
#define MAP_SIZE   100000000
#define DATA_SIZE  100000000

struct Data {
    int something;
    int key;
    int next;
};

Data data[DATA_SIZE]; //data starts at data[1]
int data_size = 0;

int hashMap[MAP_SIZE];
int map_size; //needed only for benchmarking

inline int hash(int key) { return key % map_size; }

void put(int idx) {
    int h = hash(data[idx].key);
    data[idx].next = hashMap[h];
    hashMap[h] = idx;
}
int get(int key) {
    int idx = hashMap[hash(key)];
    while (idx != 0 && data[idx].key != key)
        idx = data[idx].next;
    return idx;
}
void remove(int key) { //probably not needed
    data[get(key)].key = -1;
    //or data[idx].key = -1;
}
```



Hash map – wydajność

oś X – zapełnienie hash mapy w procentach (data_size/map_size*100%), oś Y – wielkość hash mapy (map_size), oś Z – czas [ns] jednej operacji put/get





3.Szybsze nie zawsze znaczy lepsze

- Koszt implementacji (czas implementacji)
- Wielkość danych wejściowych algorytmu
- Rekurencja jest na ogół prostsza w implementacji, a ma porównywalne wyniki wydajnościowe

4. Dynamiczna alokacja pamięci, wskaźniki – NIE!(?)

- Staramy się nie używać: new/malloc (czasem można, np. w przy grafach)
  - nadmiar: 8 bytes
  - alokator alokuje zawsze >= 32/24 bytes (uwzg. Nadm.)
- Staramy się nie używać delete/free nawet jak używamy new/malloc (chyba, że mamy test case'y)
- Wskaźnik vs. Index:
  - 8 bytes vs. 4 bytes (na 64bit maszynach)
  - problemy z debugowaniem
- Alokujemy pamięć statycznie wszędzie tam gdzie się da

5. Cache (procesora)

- CppCon 2016: Timur Doumler “Want fast C++? Know your hardware!"
- Przykład: mnożenie macierzy

```
#define MAX 2000

int A[MAX][MAX];
int B[MAX][MAX];
int C[MAX][MAX];
```

```
void mul1(int n)
{
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            for (int k = 0; k < n; k++)
                C[i][j] += A[i][k] * B[k][j];
}
```

```
void mul2(int n)
{
    for (int i = 0; i < n; i++)
            for (int k = 0; k < n; k++)
        for (int j = 0; j < n; j++)
                C[i][j] += A[i][k] * B[k][j];
}
```

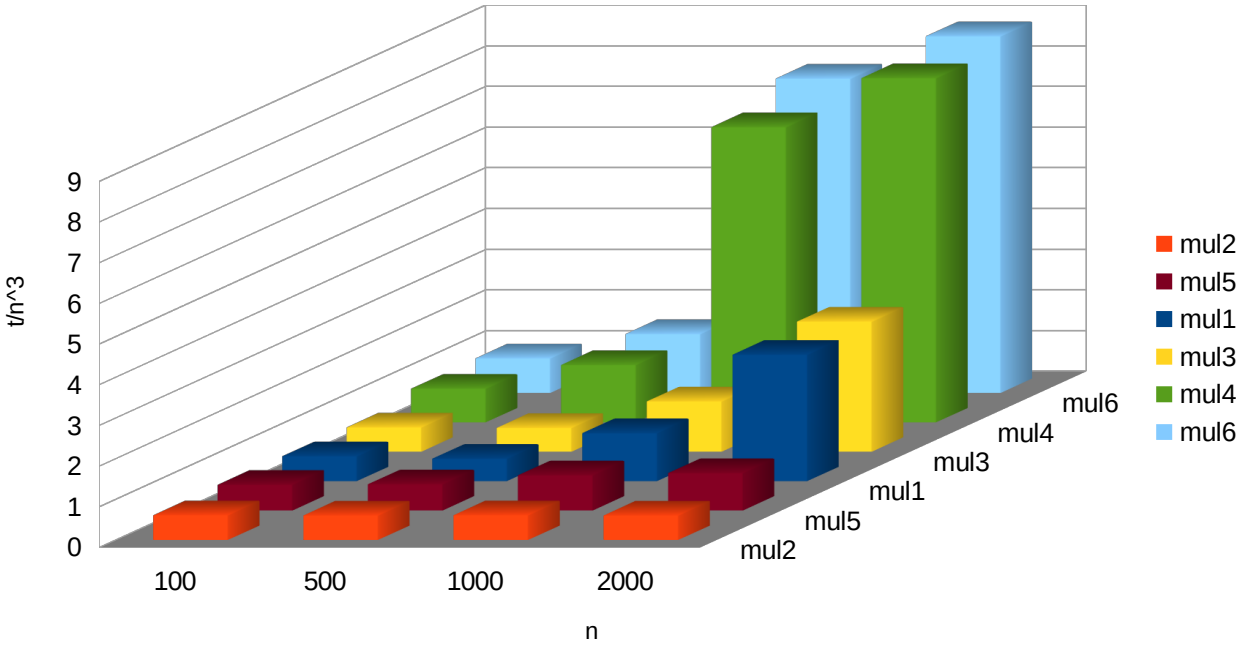
```
void mul3(int n)
{
        for (int j = 0; j < n; j++)
    for (int i = 0; i < n; i++)
            for (int k = 0; k < n; k++)
                C[i][j] += A[i][k] * B[k][j];
}
```

```
void mul4(int n)
{
        for (int j = 0; j < n; j++)
            for (int k = 0; k < n; k++)
    for (int i = 0; i < n; i++)
                C[i][j] += A[i][k] * B[k][j];
}
```

```
void mul5(int n)
{
            for (int k = 0; k < n; k++)
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
                C[i][j] += A[i][k] * B[k][j];
}
```

```
void mul6(int n)
{
            for (int k = 0; k < n; k++)
        for (int j = 0; j < n; j++)
    for (int i = 0; i < n; i++)
                C[i][j] += A[i][k] * B[k][j];
}
```

Mnożenie macierzy



n	mul1	mul2	mul3	mul4	mul5	mul6
100	0,619	0,614	0,606	0,835	0,641	0,859
500	0,558	0,608	0,594	1,419	0,652	1,457
1000	1,187	0,614	1,244	7,265	0,873	7,735
2000	3,114	0,608	3,206	8,470	0,927	8,777

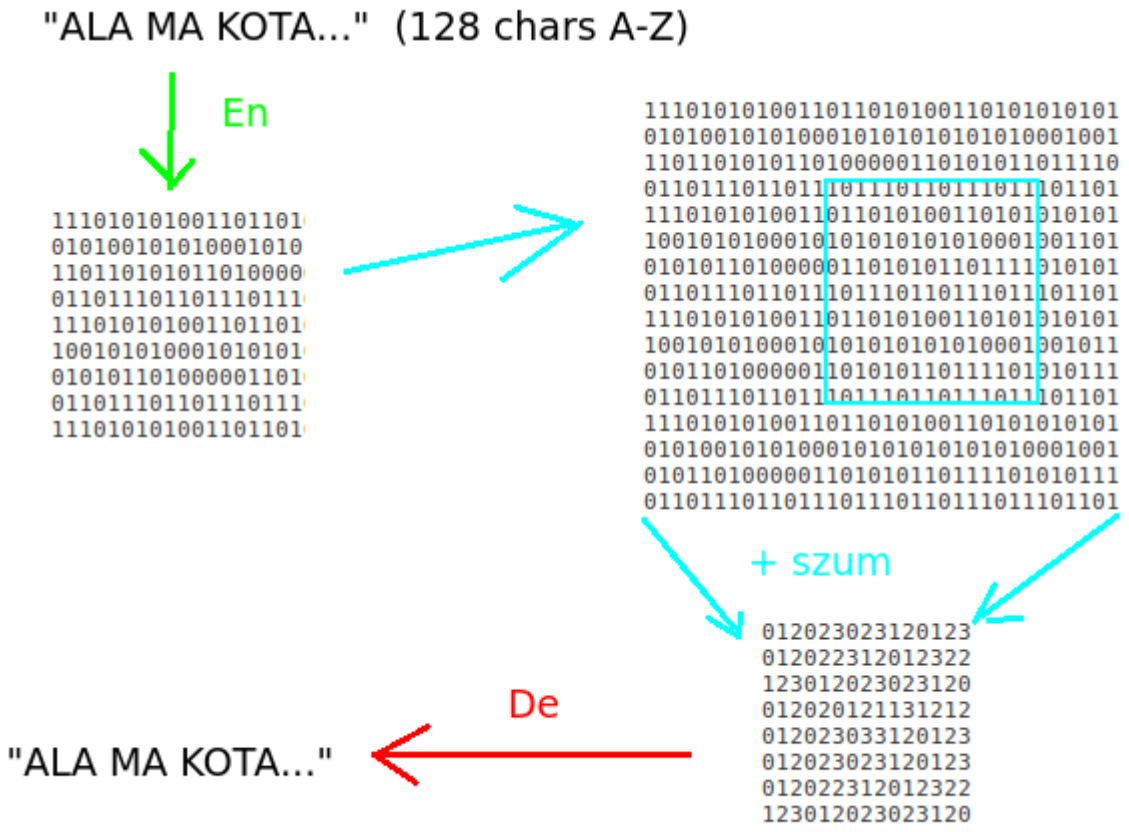
6. Omówienie przykładowych problemów

- Szybkie mnożenie
- odzyskiwanie informacji z „zaszumionych obrazków”

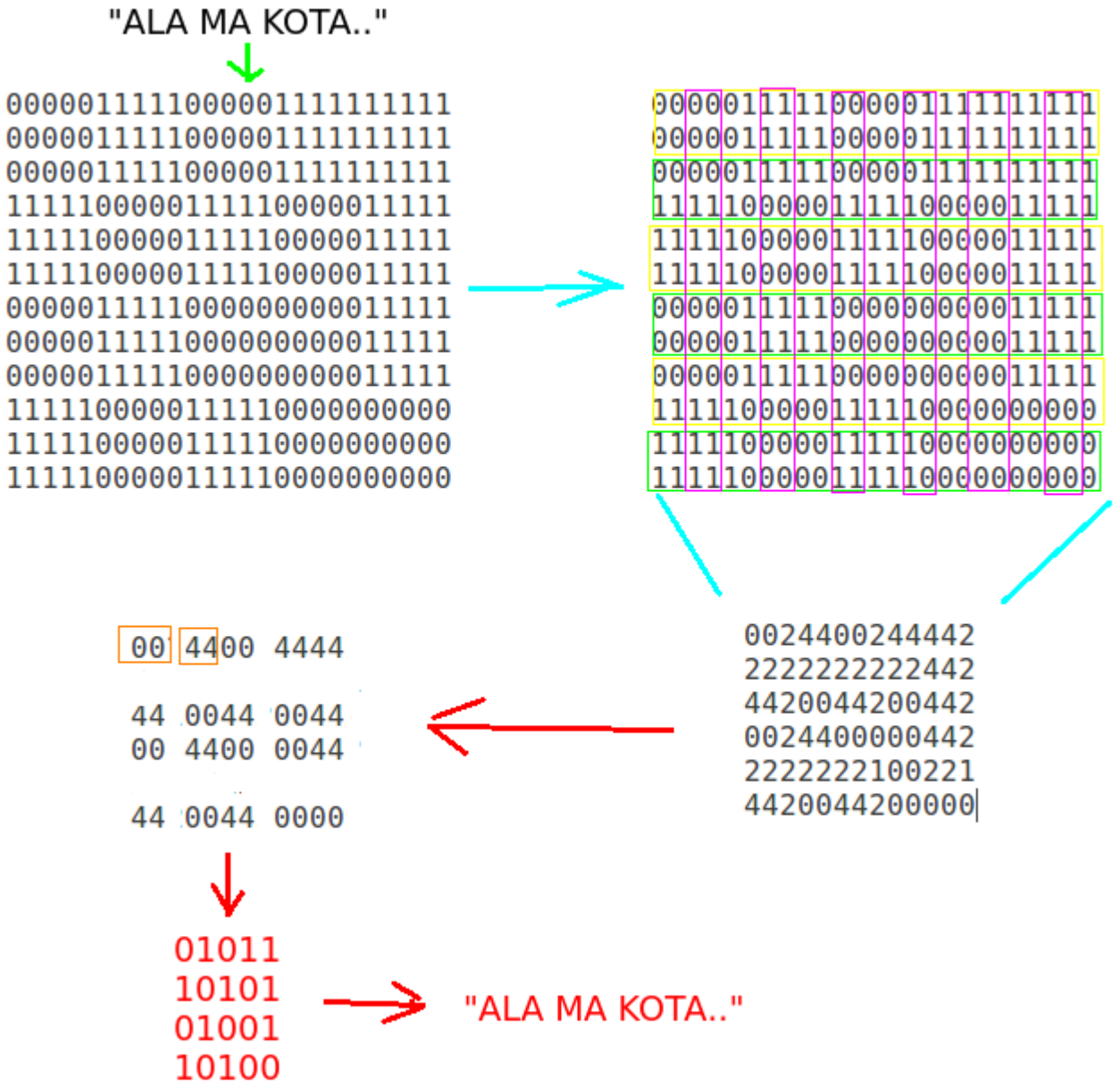
Szybkie mnożenie

```
2  #define MAX 128
3
4  #define BASE 40
5  #define DIGITS 5         //4
6  #define LEN 26           //32
7  #define LL unsigned long long
8
9  LL A[3 * LEN], B[3 * LEN], V[3 * LEN], MAX_NUM = 102400000ull;
10
11 void read(LL *v, char *str) {
12     LL m = 1;
13     for (int i = 0; i < MAX; i++) {
14         v[i / DIGITS] += str[MAX - 1 - i] * m;
15         m *= BASE;
16         if (m >= MAX_NUM) m = 1;
17     }
18 }
19
20 void write(char *str, LL *v) {
21     LL x = 0;
22     for (int i = 0; i < 2 * MAX; i++) {
23         if (i % DIGITS == 0) x += v[i / DIGITS];
24         str[2 * MAX - 1 - i] = x % BASE;
25         x /= BASE;
26     }
27 }
28
29 void mul(char *v, char *a, char *b) {
30     for (int i = 0; i < 2 * LEN; i++) B[i] = A[i] = V[i] = 0;
31
32     read(A, a);
33     read(B, b);  //0.7s
34
35     for (int i = 0; i < LEN; i++)
36         for (int j = 0; j < LEN; j++)
37             V[i + j] += A[i] * B[j];  //0.6s
38
39     write(v, V); //0.8s
40 }
```

odzyskiwanie informacji z „zaszumionych obrazków”



odzyskiwanie informacji z „zaszumionych obrazków”



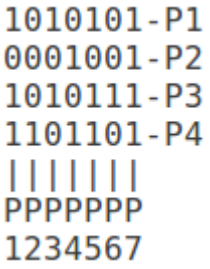
odzyskiwanie informacji z „zaszumionych obrazków”

```
/*
 * 100'000 -> 1084,5 (c-'A')
 *         -> 1071,5 (c-'A'+1)
 *         ->  910,5 (c-'A'+5)
 */
//  encode code
#define toB(c, s)   (((c)-'A'+5)<<s)
#define fromB(b)    (((b)&31)+'A'-5)
#define STR1 "PAWI"
#define STR2 "WUPE"
#define STR1_hex (toB('P',  0)|toB('A',  5)\
                  |toB('W', 10)|toB('I', 15))
#define STR2_hex (toB('W',  0)|toB('U',  5)\
                  |toB('P', 10)|toB('E', 15))
#define STR_hex(d,i) (toB(d[i],   0)|toB(d[i+1],5) \
                      |toB(d[i+2],10)|toB(d[i+3],15))

#define MASK ((1<<20)-1)
#define LL unsigned long long

void saveLL(char QRC[LEN][LEN], int &y, LL c) {
    for (int y_last = y + 3; y < y_last; y++)
        for (int x = 0; x < 5 * 20; x++)
            QRC[y][x] = (c >> (x / 5)) & 1;
}

void encode(char QRC[LEN][LEN], char SRC[LEN]) {
    char d[LEN + 8] = STR1 STR2;
    memcpy(d + 8, SRC, LEN);
    for (int y = 0, i = 0; i < LEN + 8; i += 4)
        saveLL(QRC, y, STR_hex(d, i));
}

```

```
//decode
void dToChar(LL d, char *c) {
    for (int i = 0; i < 4; i++) {
        c[i] = fromB(d);
        d >>= 5;
    }
}

LL decodeLine(char GRY[LEN][LEN], int y, int p) {
    LL d = 0, bit = 1;
    for (int i = (p + 1) / 2, j = 0; j < 40; j++) {
        if (GRY[y][i] + GRY[y][i + 1] >= 8) d |= bit;
        bit <<= 1;
        p ^= 1;
        i += 2 + (p & 1);
    }
    return d;
}

void decode(char DST[LEN], char GRY[LEN][LEN]) {
    LL out[6][LEN];

    for (int y = 0; y < LEN; y++)
        for (int p = 0; p < 5; p++)
            out[p][y] = decodeLine(GRY, y, p);

    for (int p = 0; p < 10; p++)
        for (int x = 0; x <= 20; x++)
            for (int y = 0; y <= LEN / 2; y++)
                if ( STR1_hex == ((out[p / 2][y] >> x) & MASK)
                  && STR2_hex == ((out[p / 2][y + 1
                                  + !(p & 1)] >> x) & MASK)) {
                    y = y + 1 + !(p & 1);
                    for (int i = 0; i < LEN; i += 4) {
                        p ^= 1;
                        y += 1 + !(p & 1);
                        dToChar(out[p / 2][y] >> x, &DST[i]);
                    }
                    return;
                }
}
```

odzyskiwanie informacji z „zaszumionych obrazków” ECC

- Kod Hamminga – „za trudny” w implementacji :/
- „Rectangular Parity Codes”:



Obrazki z ECC

```cpp
/*
 * 100'000 -> 4 fail
 * 100'000 -> 13 fail (without STR1,STR2)
 */
//  encode/decode code
// #define...
int parBit(UI v) {
    v ^= v >> 1; v ^= v >> 2;
    v = (v & 0x11111111U) * 0x11111111U;
    return !((v >> 28) & 1);
}

int countBits(int i) {
    i = i - ((i >> 1) & 0x55555555);
    i = (i & 0x33333333) + ((i >> 2) & 0x33333333);
    return (((i + (i >> 4)) & 0x0F0F0F0F) * 0x01010101) >> 24;
}

void saveLL(char QRC[LEN][LEN], int &y, LL c) {
    for (int y_last = y + 3; y < y_last; y++)
        for (int x = 0; x < 5 * 20; x++)
            QRC[y][x] = (c >> (x / 5)) & 1;
}

void encodeECC(char QRC[LEN][LEN], int &y, char *d, int len) {
    LL c, parH = 0, parW = 0, bit = 1;
    for (int i = 0; i < len; i += 4) {
        c = STR_hex(d, i);
        saveLL(QRC, y, c);
        parW ^= c;
        parH |= parBit(c) ? bit : 0;
        bit <<= 1;
    }
    saveLL(QRC, y, parW);
    saveLL(QRC, y, parH);
}

void encode(char QRC[LEN][LEN], char SRC[LEN]) {
    char d[LEN + 8] = STR1 STR2;
    memcpy(d + 8, SRC, LEN);
    int y = 0;
    encodeECC(QRC, y, d, 9 * 4);
    encodeECC(QRC, y, d + 9 * 4, 9 * 4);
    encodeECC(QRC, y, d + 18 * 4, 9 * 4);
}
```

```cpp
//decode
void dToChar(LL d, char *c) {
    for (int i = 0; i < 4; i++) {
        c[i] = fromB(d);
        d >>= 5;
    }
}
LL decodeLine(char GRY[LEN][LEN], int y, int p) {
    LL d = 0, bit = 1;
    int i = (p + 1) / 2;
    for (int j = 0; j < 40; j++) {
        if (GRY[y][i] + GRY[y][i + 1] >= 8) d |= bit;
        bit <<= 1; p ^= 1; i += 2 + (p & 1);
    }
    return d;
}
bool correctECC(UI *out, int len) {
    UI cor = 0, parW = 0, parH = out[len + 1];
    for (int i = 0; i <= len; i++) {
        parW ^= out[i];
    }
    for (int i = 0; i < len; i++) {
        if (parBit(out[i]) != (parH & 1))
            out[i] ^= parW, cor++;
        parH >>= 1;
    }
    return countBits(parW) + cor < 3;
}
void decode(char DST[LEN], char GRY[LEN][LEN]) {
    LL out[6][LEN];
    UI out2[LEN];
    for (int y = 0; y < LEN; y++)
        for (int p = 0; p < 5; p++)
            out[p][y] = decodeLine(GRY, y, p);

    for (int p = 0; p < 10; p++)
        for (int x = 0; x <= 20; x++)
            for (int y = 0; y <= LEN / 2; y++) {
                int y_ = y, p_ = (p & 1);
                for (int i = 0; i < 34; i++) {
                    out2[i] = (out[p / 2][y_] >> x) & MASK;
                    p_ ^= 1;
                    y_ += 1 + p_;
                }
                if ( correctECC(&out2[0],  9)
                  && out2[0] == STR1_hex
                  && out2[1] == STR2_hex
                  && correctECC(&out2[11], 9)
                  && correctECC(&out2[22], 9)) {
                    int j = 0, i;
                    for (i = 2;  j <  7 * 4; i++, j += 4)
                        dToChar(out2[i], &DST[j]);
                    for (i += 2; j < 16 * 4; i++, j += 4)
                        dToChar(out2[i], &DST[j]);
                    for (i += 2; j < 25 * 4; i++, j += 4)
                        dToChar(out2[i], &DST[j]);
                    return;
                }
            }
}
```

7. Podsumowanie i gdzie znaleźć dodatkowe materiały

- Ćwiczyć, ćwiczyć, ćwiczyć ….
- **Praca magisterska Piotra Stańczyka: „Algorytmika praktyczna w konkursach informatycznych” z 2007r.**
- Potyczki Algorytmiczne
  20 - 26 listopada

…. macro D(x)

```cpp
#include <iostream>

using namespace std;

#define D(x) x
//#define D(x)

void print(int *v) { ... }

int main()
{
    cin >> n;
    while(n--) {
        ....
        D(print(m));
        ....
        ....
        D(cout << i << ":" << j <<  " = " << r << "\t" << (((i+j)&1) && m[i*M+j]<1) << " - " << p << "\n");
        ....
        ....
        D(
            for(int i=1;i<=l;i++)
                for(int j=1;j<=l;j++)
                    print(m[i][j]);
        );
        .....
        D(cout << "-------\n");
        .....
    }
}
```

8. Pytanie / uwagi końcowe

9. Ankieta odnośnie szkolenia